VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Hra Závody aut v prostředí internetu

Game Car Race Over Internet

2015 Peter Holý

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Peter Holý**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Hra Závody aut v prostředí internetu
Game Car Race Over Internet

Zásady pro vypracování:

Cílem práce je vytvořit hru Závody aut v prostředí internetu v jazyce Java ve 2D grafice, která umožní vytváření vlastních tratí, sestavování autíček z předdefinovaných komponent a bude umožňovat připojení hráčů z počítačů a zařízení se systémem Android.

Hra bude umožňovat:
1. Výstavbu tratě.
2. Sestavování autíček z předdefinovaných komponent.
3. Závodění více hráčů.
4. Hru více hráčů.
5. Připojení prostřednictvím technologie Java WebStart.
6. Připojení ze systému Android.
7. Program umožní připojení do současně vyvíjeného portálů her v jazyce Java.

Práce bude obsahovat:
1. Implementaci hry Závody aut.
2. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.
3. Uživatelskou dokumentaci aplikace.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/
Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 7. května 2015 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2015 .....

Rád bych zde poděkoval panu Ing. Davidovi Ježkovi, Ph.D. za poskytnuté rady a své rodině a přátelům za podporu při tvorbě této práce.

Abstrakt

Tato bakalářská práce popisuje návrh a implementaci závodní hry v prostředí internetu s využitím programovacího jazyka Java ve 2D grafice. V práci je obsažena implementace hry nejen pro desktopovou platformu, ale i pro mobilní zařízení s operačním systémem Android. Součástí této práce je také návrh a implementace serverové části, která umožňuje závodění více hráčů. Dále tato práce ukazuje návrh a implementaci editoru map. Pro tvorbu grafického uživatelského rozhraní se v desktopové části jak pro hru, tak pro editor map využívá knihovna SWT.

Klíčová slova: Android, Java, Závody automobilů, Kolize, Editor map

Abstract

This bachelor thesis describes the design and implementation of racing games on the Internet using the programming language Java in 2D graphics. The work included the implementation of the game, not only for the desktop platform, but also for mobile devices running Android. Part of this work is to design and implement a server part which allows multiplayer racing. Furthermore, this work shows the design and implementation of the map editor. To create GUI in desktop part for both the game and the map editor uses the SWT library.

Keywords: Android, Java, Car racing, Collision, Map editor

Seznam použitých zkratek a symbolů

ART	–	Android Runtime
AWT	–	Abstract Window Toolkit
GUI	–	Grafické uživatelské rozhraní
HP	–	Hit Points
IBM	–	International Business Machines
IDE	–	Integrated Development Environment
IFC	–	Industry Foundation Classes
JAR	–	Java Archive
JDK	–	Java Development Kit
JNLP	–	Java Network Launching Protocol
NFC	–	Near Field Communication
OHA	–	Open Handset Alliance
OS	–	Operační systém
SDK	–	Software Development Kit
SWT	–	Standard Widget Toolkit

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Historie počítačových her se píše již od padesátých let minulého století, avšak první skutečně hratelné hry se objevily až v letech sedmdesátých. Hry textové byly jedny z prvních her na světě. V těchto hrách byl herní svět tvořen slovy, a hráč musel svoje příkazy zadávat písemně. V tomto typu her nastával problém uváznutí, kvůli nesprávně vybranému tvaru slova. Proto také vznikaly hry, které hráči umožnily zobrazit seznam všech dostupných slovních variant, které mohl použít. Hlavní výhodou těchto her byla celková nenáročnost. Jak z hlediska paměti, tak i procesoru. [1, 2]

Jednou z prvních takových her byl Star Trek. Autorem této hry je Michael Mayfield, který ji naprogramoval pro počítač Sigma 7. Hratelnost byla velmi jednoduchá. Hra spočívala v zadávání příkazů, které řídily pohyb vesmírné lodi ve vybrané části vesmíru. Další příkazy bylo možné použít pro boj s nepřátelskými loděmi, nebo například pro vypuštění raketové střely. Vybraná část vesmíru se před zadáním každého příkazu překreslila na obrazovce terminálu, který pouze dokázal vypisovat jednotlivé textové řádky. [3]

S vývojem počítačů se vyvíjely i hry samotné. Od vydání hry Star Trek v roce 1971 již uplynulo mnoho let a doba, kdy hra představovala jen text je dávno zapomenutá. Dnešní moderní počítače umožňují vytvářet moderní 2D a 3D hry, které jsou stále zdokonalovány a postupným vývojem se stávají více realistickými.

Velmi populární jsou v dnešní době hry s tématikou závodění automobilů. Tyto hry mají většinou jednoduchý princip a hratelnost samotné hry spočívá v tom, dostat své auto co nejrychleji do cíle pomocí šipek, kterými auto ovládáme. Jednou z nejznámějších takových her je Need For Speed, která vyšla pro PC v roce 1994. Od té doby byla mnohokrát zdokonalována a v roce 2013 vyšla její 20. PC verze. V letošním roce vyšla 21. verze, která je určena jen pro systémy iOS a Android, tedy pro mobilní zařízení. [4, 5]

Cílem této práce je implementace 2D multiplayer hry Závody aut v prostředí internetu, která umožní připojení hráčů z počítačů a zařízení se systémem Android. Další části práce je vytvořit editor aut a editor map, které umožní skládat vlastní automobily a tratě z předdefinovaných prvků.

2 Android OS

2.1 Historie a současnost

Společnost Android, Inc. byla založena roku 2003 a na jejím založení se podíleli Andy Rubin, Rich Miner, Nick Sears a Chris White. O dva roky později přišla společnost Google, Inc., která v té době poměrně mladý a začínající projekt zakoupila. I po jeho zakoupení zůstal ve vedení jeden ze zakládajících členů Andy Rubin. Roku 2007 byla založena tzv. OHA neboli Open Handset Alliance, která měla za úkol vytvořit standardizovaný operační systém pro mobilní zařízení. Dnes tvoří OHA více než 80 světových společností.

Výrobcem úplně prvního mobilního telefonu s Androidem byla společnost HTC v říjnu roku 2008 s názvem HTC Dream neboli T-Mobile G1. Tento mobilní telefon využíval operační systém Android 1.0 – 1.6. [6]

2.1.1 Verze Android

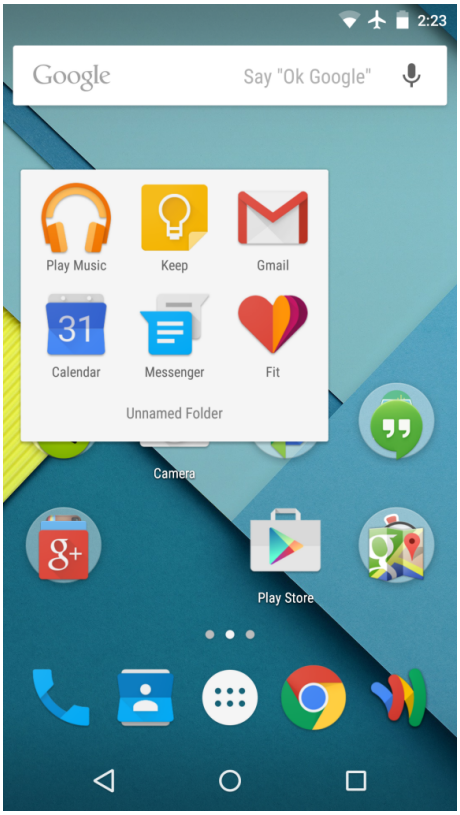
Jednou z prvních verzí která se zapsala do historie Androidu 30. dubna 2009 je verze 1.5 tzv. Cupcake [7], která měla nové widgety, umožňovala nahrávání videí a měla vylepšené funkce Copy & Paste. Následující verze vyšla 15. září 2009 a měla označení 1.6 zvaná Donut [8]. Tato verze měla vylepšený AndroidMarket, umožňovala mazat fotografie hromadně, podporovala rozlišení WVGA, vylepšovala aplikace fotoaparátu a měla některé vizuální změny.

Po ní vyšly v jednom roce 2009 hned dvě nové verze 2.0 a 2.1 s názvem Eclair [9]. Tyto verze obsahovaly vylepšené uživatelské rozhraní, nový prohlížeč s podporou pro HTML5, podporovala přisvětlovací diody, umožňovala tzv. živé tapety a celkovou optimalizaci hardwaru. Dne 20. května 2010 vyšla verze 2.2 s názvem Froyo [10], která jako první umožnila instalovat aplikace na paměťovou kartu, dále umožnila vytvoření WiFi hotspotu a měla Adobe Flash 10.1. Verze 2.3 a 2.4 zvané Gingerbread [11] vyšly také obě roku 2010 a jejich hlavní vylepšení spočívala v zásadní vizuální změně, nové podpoře NFC, podpoře protokolu SIP, vylepšené softwarové klávesnice, novou verzi Google maps a vylepšení funkce Copy & Paste.

Název Honeycomb [12] zahrnoval dokonce tři nové verze Androidu 3.0, 3.1, 3.2, které vyšly roku 2011 a byly určeny pouze pro tablety. Měly nový design optimalizovaný pro tato zařízení. Dále vylepšený multitasking, nový prohlížeč a novou funkci USB Host.

Následně vyšla 19. října 2011 verze 4.0 s názvem Ice Cream Sandwich [13], která měla nový launcher, umožňovala panoramatické focení, vylepšení správce kontaktů a měla nový ukazatel přenesených dat. Verze 4.0 se dále rozvíjela a tak roku 2012 vyšly verze 4.1 a 4.2 zvané Jelly Bean [14]. V těchto verzích byl nový Project Butter, Google Now, byla vylepšená notifikační lišta, vylepšená aplikace fotoaparátu a dále podpora více uživatelů pro tablety. Verze 4.4 s názvem KitKat [15] byla představena veřejnosti 3. září 2013. V této verzi byl odstraněn nástroj na ochranu soukromí, umožnil optimalizace pro telefony s menší pamětí RAM a dále umožnil stažení notifikační lišty v aplikacích s celoobrazovým režimem.

Android se za poslední desetiletí velmi rychle vyvíjel a tak dnešní moderní telefony s tímto systémem mají už verzi 5.0 a 5.1. tzv. Lollipop [16]. Ten má nový vzhled uživatelského rozhraní, nový efektivnější ART, nový systém správy baterií a umožňuje i tzv. mód Guest.



Obrázek 1: Plocha Android Lollipop

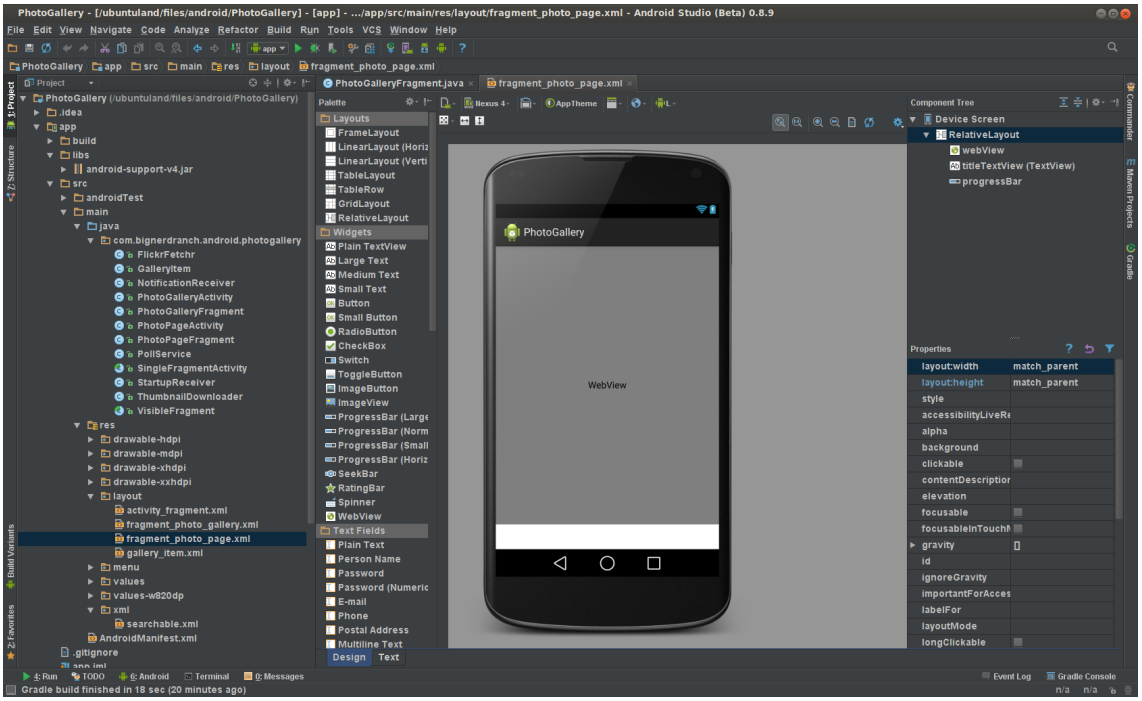
2.2 Vývoj

Android aplikace jsou psány v programovacím jazyce Java, který je jeden z nejpoužívanějších programovacích jazyků na světě, jak víme z průzkumů na serveru Indeed [17]. To je jeden z důvodů, proč se obecně Android a zvlášť jeho aplikace tak rychle vyvíjí a rozšiřují. Programovací jazyk Java vyvinula firma Sun Microsystems a představila jej 23. května 1995. Java může pracovat na různých zařízeních, například pro čipové karty je to platforma JavaCard, pro mobilní telefony JavaME, pro desktopové počítače Java SE a pro velmi rozsáhlé systémy spolupracujících počítačů Java EE. [18]

Samotná Java byla dlouhou dobu uzavřeným projektem, ale roku 2006 přišla společnost Sun Microsystems s Open Java Development Kit tzv. OpenJDK což je bezplatná open source implementace platformy Standard Edition Java SE. [19]

Abychom mohli napsaný kód správně editovat, kompilovat nebo například krokovat, potřebujeme vývojové prostředí tzv. IDE, které nám toto umožní. Jednou z možností vývojového prostředí pro kterou se můžeme rozhodnout je Eclipse, který je pro většinu lidí nejznámější. Toto vývojové prostředí bylo navrženo tak, aby bylo možné rozšiřovat seznam programovacích jazyků. Můžeme přidávat jednotlivé pluginy, např. pro C++, PHP, nebo právě pro Android. Poté stačí nainstalovat Android SDK a můžeme začít vyvíjet i s možností použití emulátoru mobilního telefonu. [20]

Další variantou vývoje Android aplikací je poměrně nové IDE s názvem Android Studio navrženo firmou Google, které bylo oficiálně představeno na konferenci Google I/O 16. května 2013. Velkou výhodou je samotná instalace, která je velice jednoduchá. Stačí stáhnout balík pro svůj operační systém a JDK pro Javu. Žádné jiné pluginy nejsou potřeba. [21]



Obrázek 2: IDE Android Studio

2.3 Vydání

Pokud již máme aplikaci hotovou a chceme ji poskytnout i jiným uživatelům systému Android, budeme se muset zaregistrovat na Google, kde nám bude automaticky vytvořen e-mail, pomocí kterého se poté můžeme přihlašovat do distribuční služby Google Play. Google Play je online služba, která vznikla 6. března 2012 spojením služeb Google Music a dřívějšího Android Marketu. Prostřednictvím této služby si můžou uživatelé z celého světa stáhnout aplikaci a záleží jen na nás, jestli naši aplikaci budeme nabízet

zdarma, nebo za poplatek. Pokud za svojí aplikaci budeme chtít poplatek, Google Play nám to umožní, ale sám si vezme 30 procent z výdělku. Pro uživatele už není nic snadnějšího než se přihlásit a stáhnout si aplikaci, musí mít ovšem přidanou platební kartu, aby za aplikaci mohl zaplatit. Pokud se uživateli aplikace nelíbí, má možnost o bezproblémové vrácení peněz do 15 minut od zakoupení. Další možnost jak získat peníze zpět je do 48 hodin od zakoupení aplikace. Nestačí však jen kliknout na vrácení peněz, protože Google požaduje vyplnit formulář, kde se uvádí důvod vrácení. [22, 23]

3 Vývoj pro Desktop

Jednou z možností programovacího jazyka pro vývoj desktopových aplikací je kromě jiných samozřejmě také Java, kterou jsem popsal v kapitole 2.2. Konkrétně v mé práci jsem použil verzi Javy 1.6. Vývojové prostředí, neboli IDE ve kterém jsem vyvíjel desktopovou aplikaci je Eclipse.

3.1 Grafické uživatelské prostředí

Pokud vyvíjíme nějakou aplikaci, musíme samotný vývoj zaměřit nejen na logickou a grafickou část, ale i propojením mezi těmito dvěma částmi. K tomu nám slouží grafické uživatelské rozhraní neboli GUI, které slouží k interakci uživatele a aplikace. Toto rozhraní je založeno na základních interaktivních grafických prvcích, které se nazývají widgety neboli komponenty. Ty mají za úkol zobrazovat informace a přijímat od uživatele akce vyvolané například kliknutím na určitou komponentu.

3.1.1 AWT

Jednou z možností, jak pracovat s GUI v Jave je použití knihovny AWT, která je původní nástroj pro tvorbu grafického uživatelského rozhraní. Knihovnu AWT vyvíjela firma Sun Microsystems jako součást Javy a byla představena s prvním představením Javy v roce 1995. Zkratka AWT, neboli Abstract Windowing Toolkit už sama o sobě vypovídá, že je tento nástroj založen na komponentách systému na kterém je spuštěn. Obstarává základní operace jako zobrazování psaného textu, zobrazování pohybu myši po monitoru, atd. Během dalšího vývoje se však ukázaly chyby v návrhu jako závislost na platformě což si odporovalo se základním konceptem Javy. Pro příklad uvedu některé základní komponenty jako jsou Canvas, Label, Button, Frame, Checkbox a Window. [24]

3.1.2 Swing

Poté co v roce 1997 Sun Microsystems upustil od vývoje AWT kvůli problémům, které nastaly a začal vyvíjet Swing. Základ Swingu je tvořen Internet Foundation Classes neboli IFC od společnosti Netscape Communications, knihovnou AWT a dalšími technologiemi. Hlavní rozdíl mezi knihovnou AWT a Swing spočívá v tom, že vzhled komponent u AWT je závislý na daném operačním systému, oproti tomu vzhled komponent Swingu je na operačním systému nezávislý. A díky tomu můžeme vzhled a chování jednotlivých komponent měnit. Swing využívá tzv. Look and Feel, což je seznam delegátů, které určují jak bude daná komponenta vypadat. Pokud tyto delegáty nenastavíme, tak budou mít výchozí vzhled, který bude na všech platformách operačního systému stejný. I zde uvedu pro příklad základní komponenty knihovny Swing což jsou JButton, JCheckBox, JComboBox, JList, JMenu nebo např. JSpinner. [25]

3.1.3 SWT

Třetí nejvíce používanou GUI knihovnou pro platformu Java je Standard Widget Toolkit neboli SWT. Tato knihovna byla vyvinuta firmou IBM a nyní společně s projektem Eclipse IDE spadá pod Eclipse Foundation. Knihovna SWT byla navržena tak, aby byla velice rychlá a výkonná. Stejně jako AWT využívá nativní knihovny operačních systémů, což znamená, že je vzhled komponent závislý na operačním systému kde je výsledný projekt spouštěn. Tato knihovna je oproti Swingu relativně jednodužší a neobsahuje žádné nadstandardní funkce, to však může některé vývojáře od SWT odrazovat. Já v mém projektu využil právě knihovnu SWT a žádný nedostatek jsem při vývoji aplikace neobjevil. [26, 27] Mezi základní komponenty patří například:

- **Browser** – prohlížeč
- **Button** – tlačítko
- **Canvas** – plátno
- **Combo** – vysouvací seznam
- **Composite** – kontejner
- **Label** – textové pole

```
package com.example.swt.widgets;
import org.eclipse.swt.widgets.*;

public class MySWTApplication {
    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setLayout(new FillLayout());

        Composite myComposite = new Composite(shell, SWT.BORDER);

        Label myLabel = new Label(myComposite, SWT.NONE);
        myLabel.setText("label");

        Button myButton = new Button(myComposite, SWT.PUSH);
        myButton.setText("button");

        shell.pack();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

Výpis 1: Vytvoření základního okna s prvky SWT

3.2 Vydání

Pokud chceme naši Java desktopovou aplikaci poskytnout i jiným uživatelům, máme možnost využití technologie Java Web Start, která nabízí jednoduchou distribuci a následné spuštění aplikace bez jakékoli instalace. První verze byla představena v březnu 2003 a nyní je součástí Java SE od verze 1.4. Samotný princip je velmi jednoduchý, stačí mít aplikaci uloženou v JAR archivu a následně na webovou stránku umístit odkaz na soubor typu JNLP, který je ve formátu XML. Ten má za úkol stažení, aktualizaci a samotné spuštění aplikace. To uživateli zaručuje, že bude mít vždy spuštěnou poslední verzi aplikace bez jakékoli aktualizace. [28]

Jediný problém s technologii Java Web Start může nastat kvůli bezpečnostnímu omezení, které povoluje spuštění pouze podepsaných JAR souborů. Toto omezení zajišťuje ochranu před nebezpečnými aplikacemi, kdy jsou uživateli při spuštění zobrazeny informace o certifikátu, kterým byl soubor JAR podepsán. A je už na uživateli samotném, jestli spuštění aplikace dovolí, nebo ne. V mé práci jsem pro testování vytvořil vlastní podpisové certifikáty, které umožňují spuštění JAR souborů. [29]

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="file:///home/applications/myApp" href="myApp.jnlp">
    <information>
        < title >Best Application</ title >
        <vendor>app</vendor>
        <homepage href="http://localhost:8080/"></homepage>
    </information>
    <version>0.1</version>
    <resources>
        <j2se version="1.6+"href="http :// java.sun.com/products/autodl/j2se"/>
        <jar href="myApp.jar"main="true" />
    </resources>
    <application-desc main-class="myApp.Main"width="300"height="300">
    </application-desc>
    <update check="background"/>
    <security >
        <all-permissions/>
    </security>
</jnlp>
```

Výpis 2: JNLP soubor se základními tagy

4 Návrh hry

Jelikož bylo úkolem mé práce vytvořit real-timovou hru, bylo nutné už v návrhu hry tuto skutečnost brát na vědomí. Pojem real-timová hra vyjadřuje typ hry, kde v reálném čase, nebo čase hrou určeným probíhá více věcí najednou a hra samotná nečeká na interakci od uživatele. Prvky hry mají předem dáno chování a pokud uživatel provede některou herní akci, hra tuto interakci příjme a zareaguje na ni.

4.1 Hlavní smyčka

Abychom mohli prvkům hry a hře samotné dát život, musíme použít hlavní smyčku hry, která se nám o to postará. Předem musíme vědět co se má v každém okamžiku aktualizace hry vlastně odehrávat. Dvě základní věci které musíme aktualizovat jsou grafická část hry, to co uživatel uvidí a logická část hry, kterou uživatel neuvidí. Tyto dvě části hry jsou navrženy odděleně a komunikují pouze prostřednictvím prvků ve hře. V mé práci bude tuto hlavní smyčku reprezentovat třída MyThread, která se v metodě run právě stará o tuto aktualizaci grafické a logické části hry. Implementaci hlavní smyčky najdete v 5.1.

4.2 Návrh logické části

Aktualizace logické části se stará o veškeré výpočty pozic a úhlů pohyblivých entit, která se nám projeví pohybem dané entity v mapě. Dále aktualizaci displeje, která se zase projeví pohybem viditelné části mapy. A v neposlední řadě testování kolizí mezi určitými kolidujícími prvky. Další podstatnou část, kterou musí logická část hry řešit, je výpočet HP daných automobilů, nebo rozhodování o vítězi hry.

4.3 Návrh grafické části

Aby byly všechny změny logické část viditelné, například pohyb automobilu do strany, je zapotřebí aktualizovat i grafickou část hry. O to se stará metoda redraw, která při jejím zavolání provede okamžité překreslení plátna a nám se tak jeví pohyb těchto entit plynulý. Aktualizace grafické části se stará například o vykreslování počtu životů daného automobilu, nebo také o vykreslování pozadí hry.

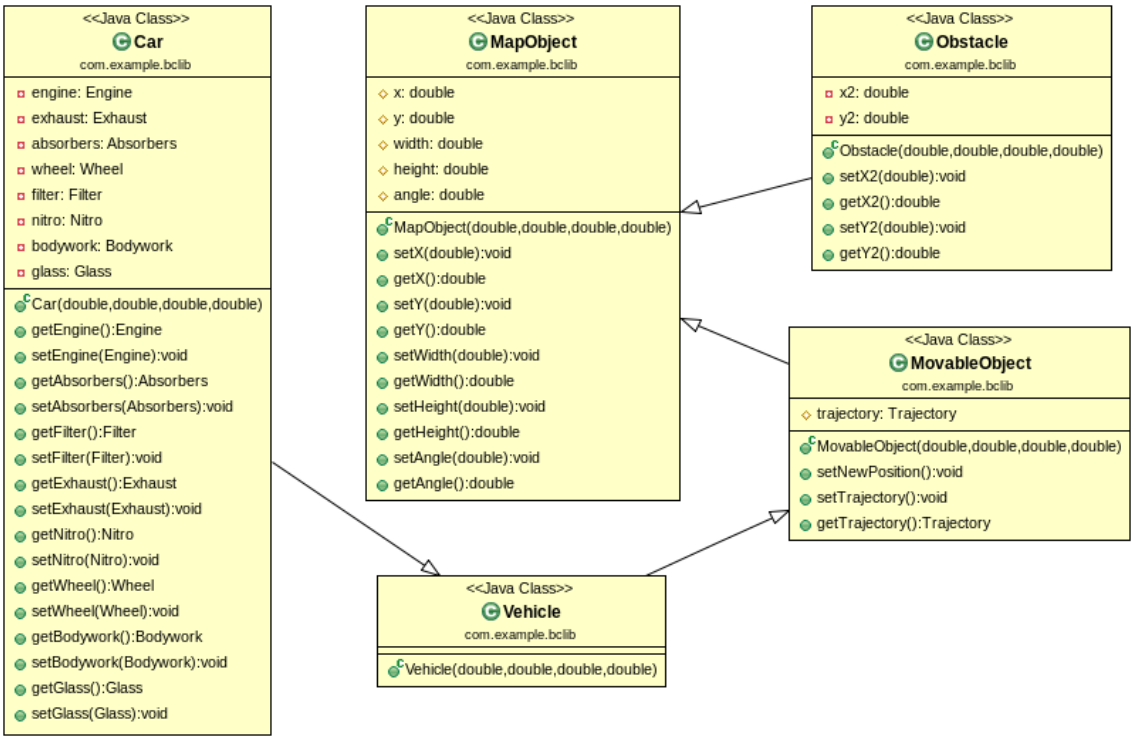
4.4 Objekty hry

Objekty, neboli prvky ve hře, jsou entity hry reprezentovány třídou MapObject, která je popsána atributy jako jsou souřadnice x, y, šířka, výška a úhel natočení. Veškeré další objekty hry budou také sdílet tyto vlastnosti a potřebné schopnosti pro práci s nimi. Tyto objekty mapy jsem si v návrhu dále rozdělil na dvě základní skupiny.

První skupina jsou prvky pohyblivé, reprezentovány třídou MovableObject, které jsou kromě základních vlastností a schopností ještě obohaceny o své vlastní, jako je vlastnost

trajektorie, reprezentována třídou Trajectory určující směr pohybu této entity a schopností setNewPosition, která nám podle dané trajektorie nastaví novou pozici. Tato třída nám ovšem pohyblivý prvek reprezentuje pouze abstraktně, ve hře se fyzicky nenachází. Následující pohyblivý prvek vycházející ze třídy MovableObject je reprezentován třídou Vehicle, který pouze říká, že se jedná o vozidlo. Také se nejedná o fyzický prvek hry. Poslední a tedy konečně fyzický prvek hry vycházející ze třídy Vehicle je reprezentován třídou Car, která má kromě zděděných vlastností a schopností, také některé své vlastní. Tato třída představuje ve hře fyzické viditelné auto, které má daný vzhled reprezentován třídami Bodywork a Glass. Výkon tohoto auta je reprezentován třídami Engine, Exhaust, Filter , Absorber, Wheel a Nitro.

Druhá skupina jsou prvky nepohyblivé, které jsou reprezentovány třídou Obstacle. Tato třída nám představuje fyzicky viditelnou překážku ve hře, která má kromě základních vlastností a schopností navíc vlastnost x2 a y2, které určují konečný bod překážky.



Obrázek 3: Diagram tříd popisující vazby

4.5 Pohyb entity v mapě

Aby se mohla každá pohyblivá entita pohybovat v mapě, musí se v každé aktualizaci měnit její x a y souřadnice. Tyto souřadnice jsou reprezentovány třídou Trajectory, která právě popisuje, o kolik se má daná entita posunout nahoru, nebo do stran. V mém návrhu hry jsem si zvolil, že se bude automobil pohybovat vždy v dané úrovni nad spodní

stranou displeje. Abychom tento fakt zajistili vždy i při rozdílné trajektorii jednotlivých automobilů, musíme aktualizovat pohyb displeje nahoru vždy právě o hodnotu kterou automobil v mapě urazil. Jelikož bylo cílem mé práce vytvořit i editor, ve kterém si uživatel může automobil poskládat nejen ze vzhledových, ale i výkonových prvků, musí se tyto změny ve výkonu projevit právě v trajektorii jeho automobilu.

4.6 Mapa hry

Mapa hry je reprezentována třídou Map a určuje nám, co se všechno v mapě fyzicky nachází. Překážky v mapě jsou řazeny v kolekci typu ArrayList podle souřadnice y. To nám pomůže při testování kolizí s těmito překážkami, u nichž se souřadnice y bude porovnávat s aktuální pozicí displeje v mapě. A jelikož budeme testovat pouze viditelné překážky, jejich seřazení nám umožní snadné procházení od začátku po konec. Jednotlivé automobily jsou také uloženy v kolekci typu ArrayList. Tato třída Map, pracující s prvky mapy, obsahuje metody, které slouží k výběru viditelných překážek na základě aktuální pozice displeje v mapě a jeho výšky. Tyto viditelné překážky jsou v každém potřebném okamžiku hry ukládány do kolekce viditelných překážek, která je použita při testování kolizí.

4.7 Testování kolizí

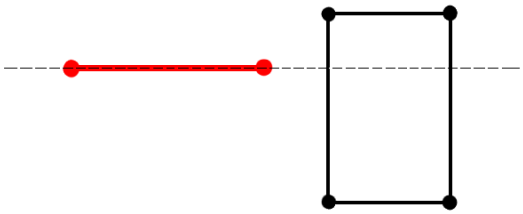
Aby nemohly jednotlivé pohyblivé entity projíždět skrze překážky a jiné entity, musí se mezi nimi testovat kolize a následně na tyto kolize nějak reagovat. K tomu nám slouží třída Collision. Ta obsahuje metodu, která slouží k testování kolizí mezi dvěmi auty. A jelikož jsou automobily vlastně obdélníky, nebude vůbec těžké otestovat, zda mezi sebou dva automobily kolidují. K tomu nám stačí otestovat, zda se jeden z rohů prvního obdélníku nachází v prostoru druhého obdélníku. A pokud takovým postupem otestujeme všechny rohy prvního obdélníku, zjistíme, zda mezi sebou dané entity kolidují, nebo ne. Abychom otestovali všechny možnosti kdy může dojít ke kolizi, musíme projít celou kolekci automobilů a následně vzájemně testovat všechny jednotlivé dvojce. Pokud se během testování detekuje mezi některou z dvojic kolize, reagujeme na ni snížením počtu životů obou automobilů a následným odrazem obou automobilů směrem od sebe.

Druhá metoda, kterou obsahuje třída Collision slouží k testování kolizí mezi automobilem a překážkou. Automobil v tomto případě můžeme také považovat za obdélník a překážku můžeme považovat za úsečku. Samotný test kolize spočívá v postupném testování všech rohů obdélníku, kde se testuje, jestli jsou rohy obdélníku nad úsečkou, pod úsečkou, nebo přímo na úsečce, což by znamenalo kolizi. Pro testování využijeme tento vzorec vycházející z rovnice úsečky.

$$F(x, y) = (y_2 - y_1)x + (x_1 - x_2)y + (x_2y_1 - x_1y_2) \tag{1}$$

Pokud je $F(x, y) = 0$, pak bod x, y leží na přímce procházející úsečkou. Pokud je $F(x, y) > 0$, pak je bod x, y umístěn nad přímkou procházející úsečkou. A v posledním případě, pokud je $F(x, y) < 0$, je bod x, y umístěn pod přímkou procházející úsečkou.

Jak vidíme na obrázku 4, úsečka je vedle obdélníku a přímka prochází skrze ní. Při tomto testování by nám vyšlo, že jsou některé body(rohy) obdélníku nad a některé pod touto úsečkou. V tomto případě je zapotřebí ještě otestovat, zda se daná úsečka náhodou nevyskytuje nad, pod, nalevo, anebo napravo od obdélníku. Pokud jeden z těchto čtyř testů výjde jako pravdivý, víme, že ke kolizi nedošlo. Pokud ovšem ke kolizi došlo, je zapotřebí na ní také reagovat, a to snížením životů u auta a následným odrazem automobilu od překážky. Implementaci kolizí najdete v 5.3.



Obrázek 4: Obdélník a přímka procházející úsečkou

4.8 Vykreslování

Další podstatnou části návrhu je vykreslování pozadí a pohybujících se entit hry. Jelikož máme pozadí hry a automobily uloženy jako obrázky, nebude složité tyto obrázky vykreslit. Musíme ale zařídit, aby se obrázky vykreslily okamžitě, když to potřebujeme a nevznikalo tak zbytečné sekání hry. Vykreslování pro desktopové aplikace a aplikace se systémem Android je velmi podobné, mají však malé rozdíly, které zde popíšu.

U desktopových aplikací máme pro vykreslování více možností, a to hlavně z důvodu více GUI knihoven, se kterými můžeme pracovat. Já jsem v mé práci použil knihovnu SWT, a tak popíšu pouze ji. Jak jsem již v kapitole 3.1.3 psal, u knihovny SWT máme jako hlavní okno aplikace třídu Display, kterou musíme vzájemně použít s plátnem pro vykreslení, což reprezentuje třída Shell. Obslužnou třídu, která obstarává vykreslování, jsem si pojmenoval SurfacePanel, a k tomu aby mohla obsahovat hlavní vykreslovací metodu paintControl, musí implementovat důležité rozhraní PaintListener. Právě tato metoda paintControl je ve vykreslování klíčová, protože se přímo v ní vykreslují jednotlivé prvky na plátno. Tato metoda je automaticky volána při aktualizaci grafické části, která probíhá v hlavní smyčce hry. Samotné provedení aktualizace nastane při zavolání metody redraw, neboli "překresli", což má za následek okamžité provedení metody paintControl.

U vykreslování pro systémy Android je nejčastější použití třídy, kterou jsem si také nazval SurfacePanel. K tomu, aby mohla obsluhovat nejen samotné vykreslování, ale též například vytvoření, změnu, nebo zničení SurfaceView plátna, musí také dědit ze třídy SurfaceView a implementovat rozhraní SurfaceHolder.Callback. To nám umožní vykreslo-

vat veškeré potřebné prvky prostřednictvím třídy Canvas. Hlavní vykreslovací metodu jsem si pojmenoval doDraw, kterou už můžeme přímo volat v hlavní smyčce při aktualizaci grafické části hry.

4.8.1 Animace ve hře

Pokud uživatel aktivuje okamžité zrychlení automobilu (nitro), vycházejí z výfuku jeho automobilu plameny. Abychom docílili dojmu skutečných plápolajících plamenů, musíme vytvořit animovaný obrázek. Obecně se při animacích zobrazují po sobě jdoucí obrázky, vždy na určitou dobu, přičemž stačí, aby bylo za jednu sekundu zobrazeno přibližně 24 po sobě jdoucích obrázků s nepatrnými rozdíly. Lidské oko tak získá dojem plynulého pohybu dané věci. Já jsem pro animaci plamene použil 24 obrázků, které však nezobrazuji postupně. Jelikož plamen šlehá nepravidelně, někdy více, někdy méně, vybírám obrázky náhodně a poté je vykresluji na danou pozici.

Obrázek 5: Snímky animace ohně

4.8.2 Vykreslování pro více rozlišení

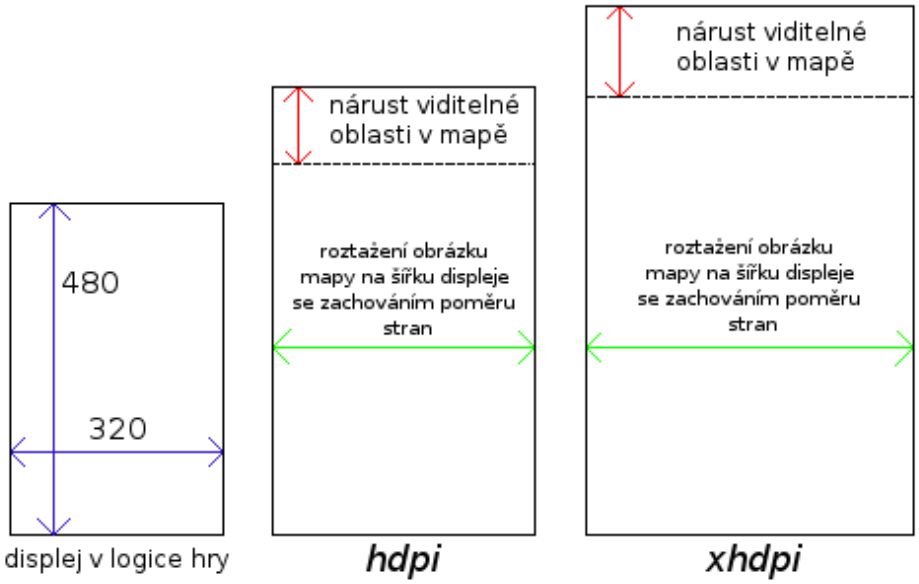
Abychom zajistili, že bude vypadat pozadí hry a jedoucí automobil stejně i na různých zařízeních, které mají jiné rozlišení displeje. Můžeme využít Androidem podporované čtyři druhy rozlišení, které nám to umožní. Tyto druhy nám představují složky, do kterých musíme nahrát obrázky v různém rozlišení. A pokud si poté načítáme obrázek z těchto zdrojů, Android nám automaticky vrátí obrázek v příslušném rozlišení, podle rozlišení daného displeje. Tyto složky jsou *ldpi*, *mdpi*, *hdpi* a *xhdpi*. Kde u *ldpi* je uváděná hodnota 120, u *mdpi* 160, u *hdpi* 240 a u *xhdpi* to je 320. Tyto hodnoty nám určují poměr mezi dp (density-independent pixel) a px (pixel). Kde u *mdpi* rozlišení to vychází, že 1 dp je roven 1 px. [30]

Problém ve vykreslování však nastává pokud nemáme obrázky hry v různém rozlišení, tak jako v mém případě. Poté však nezbývá nic jiného, než si uložit obrázky v jednom rozlišení. A následně při vykreslení přepočítávat jejich rozměry podle rozlišení daného displeje. Tímto způsobem však ještě nevyřešíme problém s vykreslováním pozadí hry, který nastává, pokud hrajeme hru na zařízeních s různým poměrem stran displeje. Tam nám nestačí pouze přepočítávat rozměry zobrazovaného pozadí podle rozměrů daného displeje. To by mělo za následek různý vzhled pozadí. Na některých zařízeních by bylo pozadí více protáhlé, nebo opačně.

V logické části hry máme třídu Display, která reprezentuje viditelnou část mapy během hry. Tento displej má předem danou výchozí hodnotu šířky a výšky. A jelikož se v mé hře pozadí zobrazuje vždy přes celou šířku displeje, stačí šířku obrázku při vykres-

lení přepočítat právě podle šířky displeje. Rozdíl bude ve výšce zobrazeného obrázku, tam pouhé přepočtení výšky obrázku podle výšky displeje nestačí.

Musí se předem zjistit rozdíl mezi výchozí hodnotou výšky displeje v logické části a hodnotou výšky displeje zařízení. Tento rozdíl se poté musí přičíst právě k výšce displeje v logické části, což má za následek, že pokud je hra spouštěna na zařízeních s různým poměrem stran, je také různá viditelná část mapy. Tedy na více podélných displejích je vidět více mapy. Tímto zajistíme, že bude pozadí mapy vypadat vždy poměrově stejně a nebude například protáhlé.



Obrázek 6: Viditelná oblast na různých rozlišeních

4.9 Interakce uživatele

Ovládání samotné hry spočívá v řízení automobilu, které jede samovolně vpřed. Uživatel tedy ovládá automobil pouze do stran, případně může použít okamžité zrychlení. Jelikož jsem u desktopové části použil knihovnu SWT, využil jsem posluchače KeyAdapter, kterého nám tato knihovna poskytuje. Tento posluchač nám umožňuje implementovat a použít události na stisknutí, nebo uvolnění tlačítka. V jednotlivých událostech už stačí testovat, které tlačítko je stisknuto, popřípadě uvolněno. Samotná implementace, tedy jak zareagujeme na uživatelovu akci, je už na nás.

U Android aplikace je zásadní rozdíl v ovládání hry, které spočívá v dotyku prstu na displeji. Při tomto dotyku můžeme zaznamenat vyvolání události onTouchEvent, ve které testujeme, zda byl displej stisknut, uvolněn, nebo byl na něm proveden pohyb. S těmito možnostmi můžeme automobil ovládat skrze vytvořená tlačítka, na kterých budeme tyto

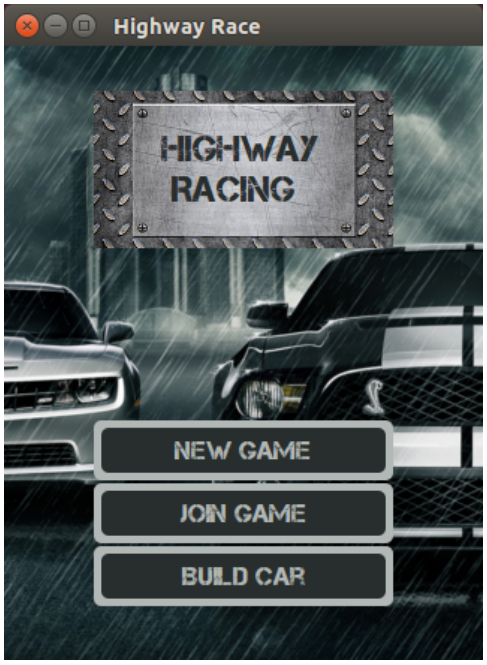
akce testovat. Testování bude spočívat ve zjištění pozice, kde se uživatel dotkl displeje. Následně se provede porovnání těchto údajů s pozicemi tlačítek v displeji.

4.10 Návrh editoru automobilů

Jeden z úkolů mého zadání je také vytvořit editor pro sestavování automobilů z předdefinovaných komponent. U tohoto editoru je verze pro desktopovou aplikaci a pro Android aplikaci téměř úplně totožná. Malé rozdíly jsou samozřejmě v použitých prvcích, které mají u těchto dvou platforem často jiný název pro totožný prvek. Práce s těmito prvky je však stejná. Tento editor je možné použít po spuštění hry, je to jedna z možností v hlavním menu. Můj editor automobilů se skládá ze dvou částí.

Tou první je část zaměřující se na vylepšení výkonu automobilu, ve které se nachází šest komponent aut. První dvě komponenty (motor a výfuk) zvyšují rychlost automobilu, tedy pohyb po ose y. Další dvě komponenty, (tlumiče a kola) zvyšují schopnost rychle ovládat automobil do stran, tedy pohyb po ose x. Následující komponenta s názvem filtr zvyšuje zrychlení automobilu po startu, které po chvíli pomine. A poslední komponenta (nitro, nebo-li tekutý dusík) zvyšuje okamžité zrychlení automobilu během závodu, které také po chvíli pomine.

Druhou částí je zaměření na vzhled automobilu, kde si uživatel vybírá ze dvou komponent automobilů. První komponenta je typ karoserie automobilu, kde má uživatel na výběr ze čtyř typů karoserií, přičemž každá karoserie je ve třech možných barvách. A druhá komponenta jsou skla automobilu, kde je možný výběr ze tří barev. Při tomto výběru je také zobrazována změna vzhledu automobilu.



Obrázek 7: Menu hry

Obrázek 8: Editor automobilů

5 Implementace hry

Součástí mé práce bylo vytvořit hru pro dvě různé platformy. Tou první je desktopová aplikace a tou druhou je Android aplikace. Z tohoto důvodu byla i implementace této hry rozdělena na dvě části. Konečným úkolem je mít hotovou hru, která bude pro obě platformy stejná. Jelikož byl vývoj pro tyto dvě různé platformy v mnoha věcech podobný, až stejný, vytvořil jsem si vlastní knihovnu tříd, kterou budou využívat obě tyto aplikace.

5.1 Implementace hlavni smyčky

Jak již jsem uváděl v návrhu hry, v real-timové hře je nezbytná hlavní smyčka hry, ve které se aktualizují jednotlivé části. Konkrétně logická část a grafická část. Pro hlavní smyčku hry jsem použil třídu pojmenovanou MyThread, která dědí ze třídy Thread. To z důvodu, aby mohla být hlavní smyčka hry vždy spuštěna na novém vlákně. Tento fakt zajistíme přepsáním metody run, kterou jsme díky třídě Thread zdědili. V následující ukázce je příklad implementace hlavní smyčky s metodou setRunning, pomocí které zvenčí ovládáme chod hlavní smyčky. Jelikož je implementace hlavní smyčky téměř totožná i pro desktopovou aplikaci, uvedu zde pouze implementaci pro Android.

```
public class MyThread extends Thread {

    private boolean mRun;
    public void setRunning(boolean boolRun) {
            mRun = boolRun;
    }

    @Override
    public void run() {
        while (mRun) {
            myCanvas = holder.lockCanvas(); //uzamceni platna, nezbytne pri zmenach

            if (myCanvas != null) {
                try {

                    myLogic.increaseValue(myGame); //aktualizace logicke casti hry
                    mySurfacePanel.doDraw(myCanvas); //aktualizace graficke casti hry

                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

                holder.unlockCanvasAndPost(myCanvas); //odemceni platna a potvrzeni zmen
            }
        }
    }
}
```

Výpis 3: Hlavní smyčka hry ve třídě MyThread

5.2 Překreslení plátna

Jelikož každá iterace hry v hlavní smyčce představuje i aktualizaci grafické části, je nutné plátno efektivně překreslovat. Jak jsem již v návrhu uvedl, u desktopových aplikací při použití knihovny SWT se při překreslení plátna metodou doDraw vyvolá okamžité zavolání metody paintControl. V této metodě se postupně volá překreslení jednotlivých komponent, jako jsou automobily a pozadí hry. U Android aplikací je překreslení téměř stejné. S tím rozdílem, že v hlavní smyčce voláme rovnou metodu doDraw, jejíž tělo je téměř identické s tělem metody paintControl. Abychom měli kód patřičně rozdělen, nebudeme v této metodě přímo provádět vykreslení. K tomu nám slouží třída Render, která se o vykreslení stará. Opět zde uvedu pouze jeden příklad této metody, a to použití u desktopové aplikace s SWT knihovnou.

```
public void paintControl(PaintEvent e) {

    myRender.draw(null, e, myShell, 0, myGame.getMapName()); //vykresleni pozadi hry

    for (Car car : myGame.getMap().getCars()) {
        myRender.draw(car, e, myShell, myGame.getMapName()); //postupne vykresleni automobilu
    }

    myRender.drawImg(e, myShell, myCar.getHp()); //vykresleni HP
}
```

Výpis 4: Metoda aktualizující grafickou část hry

5.3 Implementace kolizí

Abychom mohli přistoupit k testování kolizí mezi jednotlivými auty nebo překážkami, musíme nejprve implementovat metody tak, aby bylo dané testování rychlé. Pokud budeme testovat kolize mezi automobilem a překážkami, je zbytečné, aby se každého testování účastnily všechny překážky, i ty, u kterých je kolize nemožná.

5.3.1 Získání viditelných překážek mapy

Z tohoto důvodu jsem ve třídě Map implementoval dvě metody, které nám tento problém vyřeší. První z těchto metod s názvem getVisibleObstacles nám načte do kolekce typu ArrayList všechny viditelné překážky na základě porovnání pozice displeje v mapě a souřadnic y u překážek. I když je viditelná pouze část této překážky, je do této kolekce přidána.

```
for (Obstacle obs : obstacles) {  //prochazeni vsech prekazek

    if (obs.getY() > display.getBottom()||  //porovnani souradnice y vuci spodni strane displeje
        obs.getY2() > display.getBottom()||  //porovnani souradnice y2 vuci spodni strane displeje
        obs.getY() < display.getTop() ||  //porovnani souradnice y vuci horni strane displeje
        obs.getY2() < display.getTop()){  //porovnani souradnice y2 vuci horni strane displeje

        visibleObstacles.add(obs); //pridani prekazky mezi viditelne
    }
}
```

Výpis 5: Část metody getVisibleObstacles

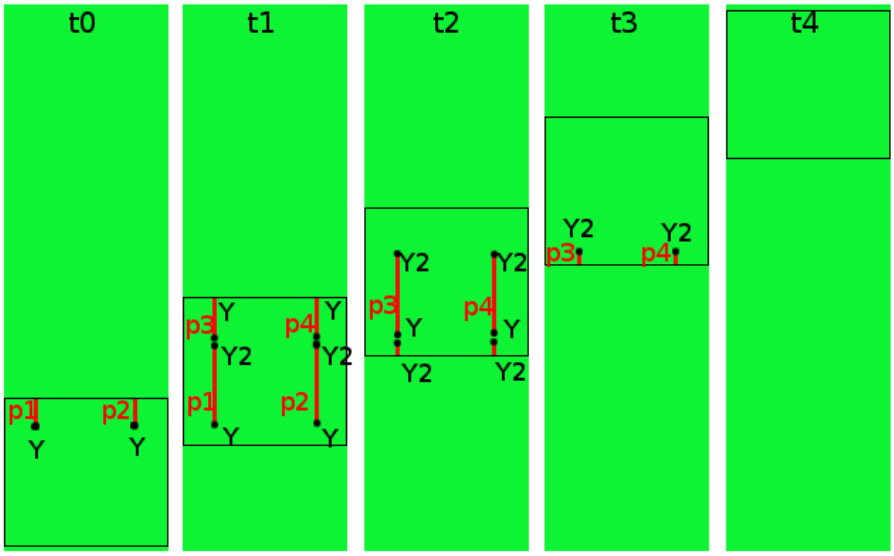
A ta druhá s názvem addAndRemoveObstacles nám při opakovaném volání tuto kolekci viditelných překážek neustále obměňuje novými. Tato metoda je rozdělena na dvě části, kde se v první části již neviditelné překážky odstraňují a v druhé části se překážky naopak přidávají do kolekce viditelných překážek. Porovnání je opět prováděno na základě pozice displeje v mapě a souřadnice y překážek. A jelikož je třída Map společná pro obě platformy, je v knihovně tříd.

```
for(int i = 0; i < visibleObstacles.size(); i++){  //prochazeni viditelnych prekazek

    if(visibleObstacles.get(i).getY2()<display.getBottom()){  //porovnani souradnice y2 a displeje

        visibleObstacles.remove(i--); //odstraneni prekazek, ktere uz nejsou videt
    }
}

int indexLast = obstacles.indexOf(visibleObstacles.get(visibleObstacles.size()-1));

for(int i = 0; i < obstacles.size() - indexLast - 1; i++){  //prochazeni prekazek

    if(obstacles.get(indexLast+1+i).getY()< display.getTop()){  //porovnani souradnice y a displeje

        visibleObstacles.add(obstacles.get(indexLast+1+i)); //pridani viditelnych prekazek
    }
}
```

Výpis 6: Část metody addAndRemoveObstacles

Jak na obrázku 9 v čase $t = 0$ vidíme, že jsou souřadnice y u překážek p1 a p2 níže, než horní strana displeje. Jsou tedy přidány do seznamu viditelných překážek. Zelená plocha v obrázku nám představuje mapu hry, černý obdélník znázorňuje displej a červené úsečky jsou překážky v mapě. V čase $t = 1$ se viditelná část displeje posunula směrem nahoru o určitou hodnotu. A kromě překážek p1 a p2, které jsou stále viditelné, se zobrazily další překážky p3 a p4, u kterých se prokázalo, že jejich y souřadnice je níže než horní strana displeje, takže jsou přidány do viditelných překážek. V čase $t = 2$ se viditelná část displeje opět posunula směrem nahoru. I když překážky p1 a p2 jsou viditelné jen z části, zůstávají v seznamu viditelných překážek. V čase $t = 3$ se pozice displeje zase aktualizovala a nám z dohledu zmizely překážky p1 a p2. Tento fakt má za násle-

dek odstranění překážek ze seznamu viditelných překážek, jelikož se potvrdilo, že jejich souřadnice $y2$ mají nižší hodnotu než je hodnota spodní strany displeje. U posledního obrázku se stejným postupem odstranily z viditelných překážek i ty s označením p3 a p4.



Obrázek 9: Viditelná oblast v čase t

5.3.2 Test kolizí s překážkami

Nyní máme k dispozici vždy jen viditelné překážky a to nám usnadní testování kolizí mezi automobilem a překážkami. A jelikož je třída Collision pro obě platformy společná, je také v knihovně tříd. Jak jsem v návrhu hry psal, testování kolizí je rozděleno na dvě části. V první je testováno, jestli jsou rohy obdélníku nad úsečkou, pod úsečkou, nebo přímo na úsečce. K tomu jsem použil vzorec vycházející z rovnice úsečky $F(x, y) = (y_2 - y_1)x + (x_1 - x_2)y + (x_2y_1 - x_1y_2)$. A jelikož musíme zaznamenat pro každý roh výsledek k pozdějšímu testování, můžeme k tomuto zaznamenání použít bitovou masku. Zaznamenávat budeme nejen pozitivní výsledky testování, kdy je roh nad překážkou, ale i negativní výsledky testování, kdy je roh pod překážkou. Abychom mohli s bitovou maskou efektivně pracovat, označíme si jednotlivé rohy číselnými hodnotami.

Roh	Hodnota	Hodnota binárně
levý horní	1	0001
pravý horní	2	0010
levý spodní	4	0100
pravý spodní	8	1000

Tabulka 1: Číselné ohodnocení rohů obdélníku

```
final int CORNER_LT = 1;
final int CORNER_RT = 2;
final int CORNER_LB = 4;
final int CORNER_RB = 8;
int RESULT_P = 0; //bitova maska pro kladne vysledky
int RESULT_N = 0; //bitova maska pro zaporne vysledky

if (obs.getPointOnLine(left, top) > 0) { // test leveho horniho rohu a prekazky
   RESULT_P = RESULT_P | CORNER_LT; //ulozeni hodnoty leveho horniho rohu
} else if (obs.getPointOnLine(left, top) < 0) {
   RESULT_N = RESULT_N | CORNER_LT;
}
if (obs.getPointOnLine(right, top) > 0) { // test praveho horniho rohu a prekazky
   RESULT_P = RESULT_P | CORNER_RT; //ulozeni hodnoty praveho horniho rohu
} else if (obs.getPointOnLine(right, top) < 0) {
   RESULT_N = RESULT_N | CORNER_RT;
}
if (obs.getPointOnLine(left, bottom) > 0) { // test leveho spodniho rohu a prekazky
   RESULT_P = RESULT_P | CORNER_LB; //ulozeni hodnoty leveho spodniho rohu
} else if (obs.getPointOnLine(left, bottom) < 0) {
   RESULT_N = RESULT_N | CORNER_LB;
}
if (obs.getPointOnLine(right, bottom) > 0) { // test praveho spodniho rohu a prekazky
   RESULT_P = RESULT_P | CORNER_RB; //ulozeni hodnoty praveho spodniho rohu
} else if (obs.getPointOnLine(right, bottom) < 0) {
   RESULT_N = RESULT_N | CORNER_RB;
}
```

Výpis 7: První část metody findingsCollision

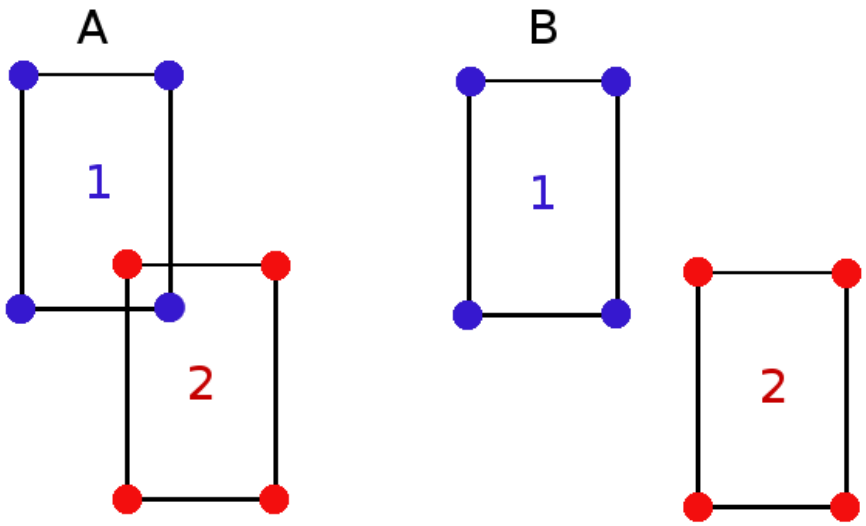
K druhé části testování dojde, jestli jsou některé body (rohy obdélníku) nad a některé pod přímkou procházející úsečkou, jako například na obrázku 4. V tomto případě je zapotřebí ještě otestovat, zda se celá úsečka nevyskytuje mimo obdélník. Jelikož používáme bitovou masku, stav kdy se všechny rohy potvrdily jako pozitivní, nebo negativní je roven číslu 15 (binárně 1111). To by však znamenalo, že jsou všechny rohy nad, nebo všechny pod obdélníkem, tedy nekolidující stav.

```
if (RESULT_P == 15 || RESULT_N == 15) return false; //nekolidujici stav
else {
   if (obs.getX() > right && obs.getX2() > right) return false; //napravo obdelniku

   if (obs.getX() < left && obs.getX2() < left) return false; //nalevo obdelniku

   if (obs.getY() > top && obs.getY2() > top) return false; //nad obdelnikem

   if (obs.getY() < bottom && obs.getY2() < bottom) return false; //pod obdelnikem

   return true;
}
```

Výpis 8: Druhá část metody findingsCollision

5.3.3 Test kolizí mezi auty

Při testování kolizí mezi automobily nám jednotlivé automobily představují obdélníky. Z tohoto důvodu jsem při implementaci metody testující kolize vytvořil třídu Point, která slouží k reprezentování bodu v mapě, konkrétně k reprezentování rohu obdélníku. Třída Point obsahuje konstruktor, který přijímá souřadnici x a y. Při použití je zapotřebí vytvořit instanci třídy Point pro každý roh obdélníku. Následný test kolize spočívá v ověření, zda jeden z rohů prvního obdélníku, není v prostoru druhého obdélníku.



Obrázek 10: Test kolize mezi auty

Jak na obrázku 10 v situaci A vidíme, ke kolizi mezi dvěma automobily došlo, protože se zjistilo, že levý horní roh automobilu s číslem 2 je v prostoru automobilu s číslem 1. Potvrdila se tedy následující podmínka, že hodnota souřadnice x tohoto červeného rohu je někde mezi hodnotou levé a pravé strany automobilu 1 a zároveň hodnota souřadnice y červeného rohu je někde mezi hodnotou horní a spodní strany automobilu s číslem 1. V situaci B se pro všechny rohy potvrdil nekolidující stav.

```
points[0] = new Point(car2.getLeft(), car2.getBottom()); //levy spodni roh
points[1] = new Point(car2.getRight(), car2.getBottom()); //pravy spodni roh
points[2] = new Point(car2.getLeft(), car2.getTop()); //levy horni roh
points[3] = new Point(car2.getRight(), car2.getTop()); //pravy horni roh

for (Point point : points) { //prochazeni vsech bodu
    if (point.getX() >= car.getLeft() && //test souradnice x a leve strany obdelniku
        point.getX() <= car.getRight() && //test souradnice x a prave strany obdelniku
        point.getY() >= car.getBottom() && //test souradnice y a spodni strany obdelniku
        point.getY() <= car.getTop()) { return true; }
}
```

Výpis 9: Část metody TestCollisionBetweenCars

5.3.4 Rotace bodu

Abychom mohli testovat automobily a překážky, které nejsou jen ve vertikální, nebo horizontální poloze, implementoval jsem metodu rotate, která umožní i testování kolizí mezi dvěma automobily s určitým úhlem. Metoda je ve třídě Point a využívá rotační matici k rotování bodu x, y podle zvoleného úhlu α. Tyto rotované body nám představují rohy automobilu a protože se automobil nachází na určité pozici, musíme rotaci těchto rohů provést na základě jeho středu c_x, c_y. Abychom mohli bod rotovat podle středu automobilu, je nutné od souřadnic rohu odečíst souřadnice středu automobilu, což musí být zakomponováno do naší rotační matice. Po provedení rotace se musí rotovaný bod opět vrátit na svou původní pozici a tak se musí k jeho souřadnicím přičíst hodnoty souřadnic středu automobilu. Rotace tedy bude podle následujícího vzorce. [31]

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \end{bmatrix} \right) + \begin{bmatrix} c_x \\ c_y \end{bmatrix} \tag{2}$$

5.3.5 Vektor

Pokud ke kolizi mezi automobilem a překážkou dojde, je nutné na ni nějak zareagovat. V mém případě jsem se rozhodl kolidující automobil od překážky odrazit pod úhlem, ve kterém narazil. Tedy úhel dopadu se rovná úhlu odrazu. Stejný postup použijeme, pokud nastane kolize mezi dvěma automobily. A jelikož platí, že pokud první automobil narazí do druhého, tak i druhý automobil narazí do prvního, je zřejmé, že se automobily od sebe odrazí pod úhly, ve kterých došlo ke kolizi.

Z tohoto důvodu jsem implementoval třídu Vector, která slouží k reprezentování automobilu a překážky v podobě 2D vektoru. Takový vektor nám usnadní práci s určováním výsledného směru odrazu. Abychom vyjádřili překážku v podobě vektoru, stačí provést odečtení souřadnice x_2 od souřadnice x_1 a to samé platí pro souřadnici y, tedy odečíst y_2 od souřadnice y_1. Tímto odečtením zjistíme velikost nového vektoru. První parametr konstruktoru třídy Vector nám představuje souřadnici x a druhý parametr představuje souřadnici y.

```
new Vector(obstacle.getX2() − obstacle.getX(), obstacle.getY2() − obstacle.getY());
```

Výpis 10: Vytvoření vektoru na základě překážky

Vyjádření vektoru na základě automobilu je ovšem složitější. Zde nestačí odečíst souřadnice x a y, musí se do výpočtu zakomponovat šířka automobilu (delší strana) a úhel automobilu. Novou souřadnici x vektoru vyjádříme pomocí funkce $\cos$ úhlu automobilu, kterou musíme vynásobit šířkou automobilu. Ta udává kolikrát bude vektor větší. Souřadnici y vyjádříme naopak pomocí funkce $\sin$ úhlu automobilu, kterou také násobíme šířkou kvůli velikosti vektoru.

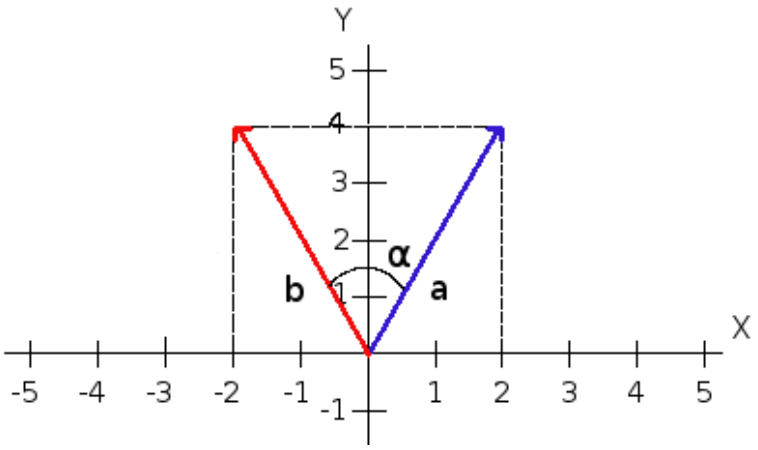
```
return new Vector(car.width ∗ Math.cos(car.angle), car.width ∗ Math.sin(car.angle));
```

Výpis 11: Vytvoření vektoru na základě automobilu

5.3.6 Směr odrazu

Vytvořený vektor z překážky a automobilu nám ulehčí zjištění výsledného směru odrazu a také hodnotu nového úhlu pro automobil. Pokud máme vektor a reprezentující překážku a vektor b reprezentující automobil, musíme zjistit jaký úhel mezi sebou svírají. Ten zjistíme pomocí skalárního součinu dvou vektorů.

$$\cos\alpha = \frac{a \cdot b}{\|a\|\,\|b\|} = \frac{a_x b_x + a_y b_y}{\sqrt{a_x^2 + a_y^2}\sqrt{b_x^2 + b_y^2}} \tag{3}$$



Obrázek 11: Skalární součin vektorů

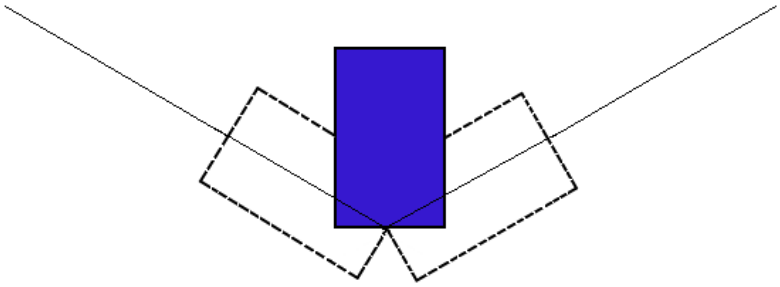
Poté musíme ještě zjistit z jaké strany automobil do překážky narazil a podle tohoto zjištění s vypočteným úhlem dále pracovat. Pro zjištění této strany použijeme vektorový součin, který se běžně používá pro 3D vektory. Nás však bude zajímat pouze jeho z souřadnice, která nám určí směr podle pravidla pravé ruky. Pro výpočet třetí souřadnice vektorového součinu použijeme vzorec $z = a_x b_y - a_y b_x$, pomocí kterého zjistíme výsledný směr odrazu. Platí zde, že pokud je výsledek menší než 0, bude směr odrazu roven úhlu mezi automobilem a překážkou, který se ještě musí sečíst s úhlem překážky. A pokud bude výsledek větší než 0, bude směr odrazu roven, ne součtu, ale rozdílu těchto úhlů.

```
Vector v1 = v.getVectorByObstacle(o); //ziskani vektoru na zaklade prekazky
Vector v2 = v.getVectorByCar(car); //ziskani vektoru na zaklade automobilu

double angle = v1.getAngle(v2); // zjisteni uhlu mezi automobilem a prekazkou

//vypocet smeru podle vzorce z = ax by−ay bx
double smer = v1.getX() * v2.getY() − v1.getY() * v2.getX();

//vysledne porovnani
if (smer < 0) { return o.getAngle() + angle; } //vraceni vysledku
else { return o.getAngle() − angle; } //vraceni vysledku
```

Výpis 12: Část metody TestCollision

5.4 Pohyb automobilu

Pokud hráč stiskne tlačítko doprava, nebo doleva, mění tím úhel svého automobilu o určitou hodnotu. A jelikož se při každé aktualizaci pozice mění nejen souřadnice, ale i úhel, je dobré tento úhel zakomponovat do výpočtu nových souřadnic. Tím docílíme, že automobil pojede rovně ve směru úhlu, tak jako v realitě. Aby byla hra dobře hratelná, určil jsem, že každý automobil může zatáčet pouze v určitém rozmezí úhlu, jak znázorňuje následující obrázek. Při aktualizaci je tedy nutné otestovat, zda se nově příchozí úhel vůbec může automobilu nastavit. Zjistí se, jestli je menší než maximální hodnota v rozmezí a zároveň jestli je větší než minimální hodnota v povoleném rozmezí. Pokud tuto podmínku splňuje, nastaví se jeho hodnota do proměnné určující úhel automobilu. Pokud tato podmínka není splněna, vyplývá z toho, že aktuální úhel automobilu je nastaven na maximum a tak se tedy nemění.



Obrázek 12: Rozmezí povolených úhlů

Pro nastavení nové pozice jsem implementoval třídu Trajectory, která obsahuje členské proměnné x, y a metody getX a getY. Tyto proměnné mají pevně danou hodnotu, ze které vychází pohyb pro všechny automobily. Tyto metody nám pro každou souřadnici vrátí hodnotu, která se k aktuální hodnotě souřadnice automobilu přičte a tím se zajistí pohyb automobilu v mapě. Přičemž hodnota, která nám je z metod getX a getY vrácena, není vždy stejná. Závisí na aktuálním úhlu automobilu. Jelikož je námi definovaná trajektorie automobilu 2D vektor, výpočet přírůstku bude podobný jako při vytváření vektoru reprezentující automobil. Jak jsem už psal v návrhu, cílem mé práce bylo také vytvořit editor automobilů, ve kterém se vybrané součástky projeví právě na trajektorii automobilu. V mém případě je to pak procentuální přírůstek podle hodnoty zvolené součástky.

```
private int speed = 7;

public double getX(double angle) {

    //vypocet prirustku  na zaklade uhlu a pevne hodnoty x
    return this.speed * Math.cos(angle);
}

public double getY(double angle) {

    //vypocet prirustku  na zaklade uhlu a pevne hodnoty y
    return this.speed * Math.sin(angle);
}
```

Výpis 13: Metody třídy Trajectory

6 Návrh multiplayer

Součástí mé práce bylo také umožnění hry pro více hráčů. Musel jsem již v návrhu některé herní části rozdělit podle toho, kde se budou vykonávat. V mém případě je pro hru více hráčů nutně spuštěn server, který se stará o obsluhu požadavků klienta a také o hru samotnou, která na tomto serveru běží.

6.1 Návrh z pohledu serveru

Jelikož server nebude obstarávat pouze chod hry, ale i komunikaci s jednotlivými klienty, kteří se na něj připojí, je nutné, aby uměl zpracovat velké množství požadavků. Požadavky klientů jsou serveru zasílány při určitých akcích hráče, jako je například vytvoření nové hry, nebo připojení do vytvořené hry. Také samotné připojení je provedeno při těchto akcích. Po připojení do hry a následném odehraní je důležité hru ukončit, aby se server zbytečně nezatěžoval již ukončenými hrami. Další důležitým úkolem serverové části je zprostředkování možných map pro hru, které jsou poskytnuty klientovi.

6.2 Návrh z pohledu klienta

Klientovi je umožněno vytvoření hry, zjištění vytvořených her a připojení se do vytvořené hry. Pokud se klient rozhodne vytvořit hru, je mu ze serveru zaslán seznam všech možných map, které server nabízí. Při výběru některé z nich a zvolení počtu hráčů ve hře, je zjištěno, zda hráč tuto mapu již má uloženou anebo ne. Pokud ji nemá uloženou, je hráči tato mapa zaslána a poté se čeká na připojení ostatních hráčů. Pokud se ostatní hráči připojí do již vytvořené hry, také se zjistí, zda mají mapu pro danou hru už uloženou, nebo ne.

6.3 Komunikace

Komunikace mezi klientem a serverem probíhá na určitém portu a IP adrese serveru, kterou musí klienti znát. Jednotlivá data, ať už to jsou čísla nebo znaky, jsou mezi klientem a serverem posílány prostřednictvím streamů, které se dělí na dva základní typy. Je to OutputStream, neboli odchozí proud a InputStream, neboli příchozí proud. Ty získáme ze třídy ServerSocket, u které se právě používá námi zvolený port.

Abychom při komunikaci mezi klientem a serverem nemuseli pokaždé porovnávat řetězce představující jednotlivé požadavky, vytvořil jsem pro velmi časté požadavky třídu CommandClass, ve které budou jednotlivé požadavky reprezentovány číselnými kódy. Poté stačí při běžné, nebo herní komunikaci využívat tuto třídu, což vede k urychlení komunikace. Obě strany, jak server, tak klient při komunikaci už jen porovnávají hodnoty těchto číselných kódů. Mezi základní příkazy patří:

- **cmdCreate = 6** / *vytvoření hry*
- **cmdJoin = 7** / *připojení do hry*
- **cmdGetGame = 8** / *zjištění vytvořených her*

- **cmdLeftPush = 10** / *odbočení do leva*
- **cmdRightPush = 11** / *odbočení do prava*

6.4 Hlavní smyčka serveru

Aby mohl server neustále přijímat a zpracovávat požadavky klienta, je nutné, aby jeho činnost neustále běžela ve smyčce. Ta je ve třídě Main a slouží pouze ke zpracování požadavků, ne však k samotné hře. Je rozdělena do několika částí, ve kterých se zpracovávají určité herní požadavky. Při komunikaci s klientem se nejprve daný požadavek načte a následně se testuje číselná hodnota tohoto požadavku. Pokud se například prokáže, že hodnota příchozího požadavku se rovná hodnotě požadavku cmdCreate, je tento požadavek zpracován a hra je vytvořena. Implementaci hlavní smyčky serveru najdete v 7.1.1.

6.4.1 Vytvoření hry

Jednou z hlavních částí je vytvoření hry odpovídající příkazu cmdCreate, ve které se od klienta přijmou důležité parametry, jako je počet hráčů a název mapy. Za pomocí těchto parametrů je vytvořena hra reprezentována třídou Game. Následně je vytvořen hráč reprezentovaný třídou Player, kterému se také nastaví důležité parametry, které přijal server od klienta. Mezi ně patří například vzhled zvoleného automobilu, jednotlivé výkonové prvky automobilu a rozměry displeje, které odpovídají hráčově viditelné oblasti. Po nastavení všech těchto parametrů je hráč přidán do hry. Následně je tato vytvořená hra přidána do seznamu vytvořených her. Poté se otestuje, zda jsou všichni hráči připojeni a pokud ano, je hra odstartována.

6.4.2 Připojení do hry

Připojení do hry odpovídající příkazu cmdJoin je velmi podobné. S tím rozdílem, že se nová hra nemusí vytvářet. Klient se připojuje do již vytvořené hry a tak tedy stačí, když zašle na server ID vybrané hry. Podle ID se tato hra najde v seznamu vytvořených her. Při připojení jsou také přijímány od klienta veškeré podstatné parametry potřebné k vytvoření hráče jako v minulém případě, tedy vzhled zvoleného automobilu, jednotlivé výkonové prvky automobilu a rozměry displeje. Hráč je poté do vytvořené hry přidán a opět nastává testování přítomnosti všech hráčů. Pokud jsou všichni připojení, je hra spuštěna.

6.4.3 Další požadavky klienta

Vytvoření a připojení do hry však nestačí. Hlavní smyčka serveru musí také obsluhovat požadavky klienta jako je získání ID všech vytvořených her, které se uživateli zobrazí při volbě připojení do hry. Dále získání názvů všech dostupných map, které se uživateli zobrazí při vytvoření hry. Také získání konkretního názvu mapy, která se v dané hře hraje. A v neposlední řadě požadavek na stažení konkrétní mapy ze serveru.

6.5 Herní smyčka na serveru

Abychom jednotlivým klientům ulehčili veškerou práci s herní logikou, která obstarává veškeré výpočty a testování, je tato herní logika přesunuta právě na serverovou část. Klienti se poté musí starat pouze o získávání informací ze serveru a samotné vykreslování hry. Tímto rozdělením také zařídíme, že můžeme ze serveru řídit tok informací my a jednotliví hráči pak budou ve stejné úrovni hry, což je při multiplayer hře nezbytné.

Hlavní herní smyčka je ve třídě Game, která reprezentuje hru. Je to cyklus, který se opakuje po celou dobu hry, respektive po dobu dokud je alespoň jeden hráč ve hře připojen. Samotná herní smyčka představuje obsluhu hráčů, kde každý hráč má svůj automobil, se kterým se pracuje. Tato obsluha znamená postupné a opakované procházení všech připojených hráčů. Jako první se v této smyčce hráčovu automobilu nastaví pozice a úhel. Dále se provádí testování, jestli některý z hráčů dojel do cíle. Následně je provedena aktualizace displeje, která se provádí směrem nahoru, tedy po ose y o hodnotu, kterou urazilo auto během jedné aktualizace. To nám zajistí, že bude automobil vždy ve stejné úrovni displeje. Tato aktualizace displeje se provádí pouze kvůli posunutí viditelné oblasti, která je klíčová při testování kolizí. Dále se testují zmíněné kolize, a to mezi automobilem a překážkami. A jako poslední se testují kolize mezi jednotlivými auty. Všechny tyto logické operace jsou prováděny postupně nad všemi automobily všech hráčů. Implementaci herní smyčky najdete v 7.1.2.

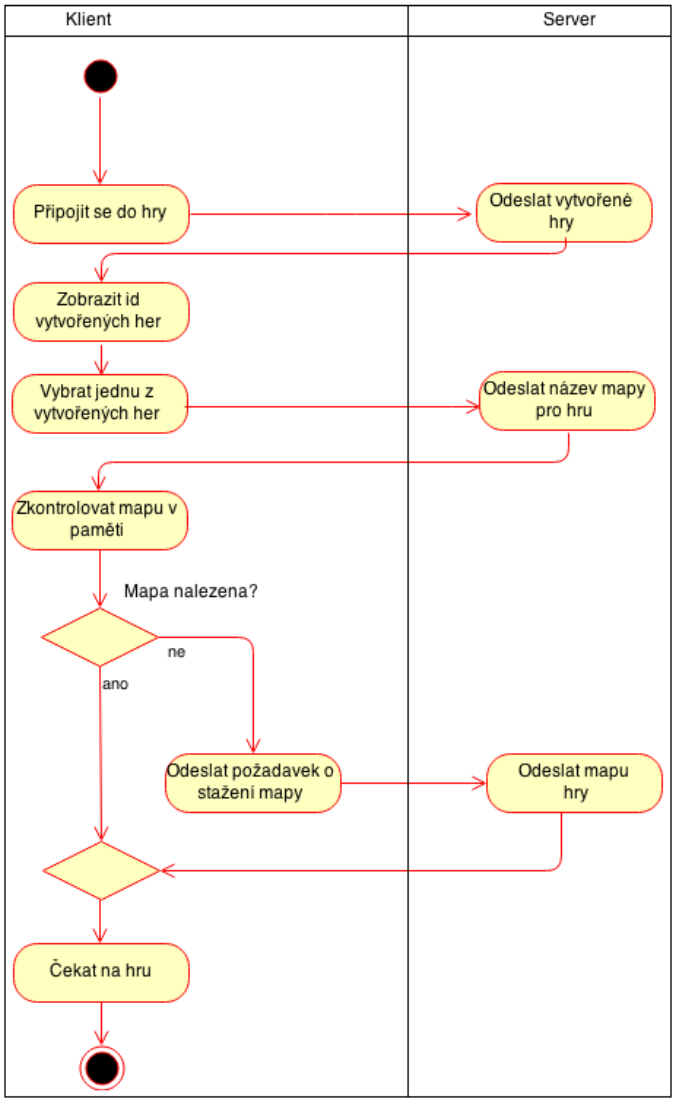
6.6 Hráčova smyčka na serveru

Klient je na serveru reprezentovaný třídou Player a jak už jsem uvedl, každý hráč má na serveru svůj automobil a displej. U automobilu se aktualizují určité parametry, jako jsou souřadnice, nebo například úhel automobilu. Je ovšem nutné, aby uživatel, tedy hráč, s tímto serverem vedl komunikaci, ve které si zase zpětně aktualizovaná data stáhne. Aby mohl každý klient neustále komunikovat se serverem, je pro jeho komunikaci také použita smyčka. V této smyčce je využita třída CommandClass, která je využívaná při běžné herní komunikaci. Pokud chce například hráč jet se svým automobilem doleva, stiskne šipku na klávesnici doleva, což u něj vyvolá událost, ve které se provede odeslání příkazu cmdLeft na server. A jelikož je komunikace ve smyčce, hráč na serveru neustále čeká na nově příchozí příkaz. Příjde mu tedy příkaz cmdLeft, kterému odpovídá určitý číselný kód a tím se nastaví hráčovu automobilu nová souřadnice a úhel odpovídající zabočení doleva. Klient ovšem potřebuje aktualizované informace ze serveru stahovat. K tomu slouží příkaz cmdGetPoses, který mu postupně pošle veškeré potřebné informace o všech automobilech ve hře a tak může nejen své, ale i soupeřovy automobily vykreslit na správných aktualizovaných pozicích. Implementaci hráčovy smyčky najdete v 7.1.3.

6.7 Ukončení komunikace

Pokud klient dojel do cíle a ukončil hru, nebo ukončil hru během závodu, je serverové části, konkrétně hráči, odeslán příkaz cmdCloseSocket na ukončení komunikace, který způsobí ukončení komunikační smyčky hráče na serveru. Také před zahájením každé

herní iterace je ve smyčce hry prováděn test, jestli je ve hře vůbec ještě někdo připojen. A jelikož se hráči mohou odpojit i během hry, je hra dohrána i kdyby v ní jel jen jeden poslední připojený hráč. Pokud ovšem není nikdo připojen, je tato herní smyčka ukončena. To má za následek odstranění hry ze seznamu vytvořených her a následné ukončení vlákna pro hru.



Obrázek 13: Diagram aktivit popisující připojení do hry

7 Implementace multiplayer

Před samotným začátkem implementace multiplayer hry, je důležité si uvědomit, že komunikace mezi klientem a serverem nesmí být nijak omezena například počtem příchozích požadavků. Dále také nesmí být omezen počet možných komunikujících klientů s tímto serverem. Pro vývoj jsem použil také programovací jazyk Java SE verze 1.6.

7.1 Serverová část

Při implementaci serverové části, jsem se zaměřil hlavně na to, aby na něm bylo možné hrát více založených her najednou s libovolným počtem hráčů ve hře. Z tohoto důvodu je nutné použít vlákna, aby chod hry a komunikace s klienty běžely současně na samostatných vláknech.

7.1.1 Hlavní smyčka serveru

Hlavní smyčka serveru je ve třídě Main a probíhá ve statické metodě main. Tato třída obsahuje seznam vytvořených her, který je uložen v kolekci typu ArrayList. Pro hlavní smyčku jsem použil cyklus **while**, kde je v jeho podmínce proměnná serverRun určující chod smyčky serveru. Tato proměnná je primárně nastavena na hodnotu **true**, tedy hodnotu, při které se smyčka opakuje. Celý kód hlavní smyčky musí být obalen do bloku **try**/**catch** kvůli případným komunikačním problémům. Po přijetí požadavku o komunikaci mezi klientem a serverem jsem pro posílání dat použil DataInputStream a DataOutputStream. Jednotlivé zpracování požadavků je rozděleno, podle typu požadavku.

```
while (serverRun) { //smycka serveru
    try {
        final Socket s = server.accept(); //komunikacni socket na urcitem portu
        final DataInputStream dis = new DataInputStream(s.getInputStream());
        final DataOutputStream dos = new DataOutputStream(s.getOutputStream());

        int command = is.read(); //nacteni pozadavku
        if (command == CommandClass.cmdCreate) {
            //zpracovani pozadavku na vytvoreni hry
        } else if (command == CommandClass.cmdJoin) {
            //zpracovani pozadavku na pripojeni do hry
        } else if (command == CommandClass.cmdGetGames) {
            //zpracovani pozadavku pro zjisteni vytvorenych her
        } else if (command == CommandClass.cmdLoadMap) {
            //zpracovani pozadavku pro stazeni mapy
        } else if (command == CommandClass.cmdGetMaps) {
            //zpracovani pozadavku pro zjisteni vytvorenych map
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Výpis 14: Kostra smyčky serveru

Základní parametry při zpracování požadavku na vytvoření hry jsou:

1. vytvoření objektu hry
2. přijetí důležitých parametrů od klienta
3. nastavení zvolené mapy a počtu hráčů pro hru
4. vytvoření objektu hráče
5. přidání hráče do hry
6. načtení logických částí mapy pro hru
7. přidání hry do seznamu vytvořených her

Kromě těchto parametrů, se musí v hlavní smyčce při vytvoření hry testovat, zda jsou všichni hráči ke hře připojeni. Toto testování představuje neustálé ověřování, kde se porovnává zadaný počet hráčů pro hru s aktuálním počtem připojených hráčů. To probíhá zvlášť na nově vytvořeném vlákně gameThread ve smyčce. A to z důvodu, aby nebyl provoz serveru tímto čekáním na hráče blokován. Pokud je potvrzeno připojení všech hráčů do hry, je provedena synchronizace startu, která představuje odeslání signálu všem hráčům. Dále je odstartována metoda run na druhém novém vlákně playerThread, která slouží ke komunikaci hráče během hry. Jako poslední je odstartována hlavní smyčka hry stejnojmennou metodou run. Pokud hra skončí a ukončí se její hlavní smyčka je v tomto herním vláknu ještě zapotřebí hru odstranit ze seznamu vytvořených her.

```
final Thread playerThread = new Thread(new Runnable() { //hracovo vlakno
    @Override
    public void run() {
        player.run(); //spusteni komunikacni smycky hrace
    }
});
final Thread gameThread = new Thread(new Runnable() { //herni vlakno
    @Override
    public void run() {
        while(game.getCountPlay()!=game.getCountPlayers()){ //zjisteni pripojenych hracu
            try {
                Thread.sleep(100); //uspani vlakna pro snizeni zatizeni
            } catch (InterruptedException e) { e.printStackTrace(); }
        }
        try {
            dos.writeBoolean(true); //odeslani signalu pro synchronizaci startu
        } catch (IOException e) { e.printStackTrace(); }
        playerThread.start(); //odstartovani hracova vlakna
        game.run(); //odstartovani herni smycky
        createdGame.remove(game); //odstraneni hry ze seznamu vytvorenych her
    }
});
gameThread.start(); // odstartovani herniho vlakna
```

Výpis 15: Vlákna ve smyčce serveru

7.1.2 Smyčka hry na serveru

Logika hry je umístěna v metodě run třídy Game. Pro tuto smyčku jsem použil cyklus **while**, kde je v jeho podmínce proměnná gameRun určující chod hry. Tato třída obsahuje seznam všech hráčů ve hře, kteří jsou uloženi v kolekci typu ArrayList. Jelikož mají klienti za úkol získávat informace o automobilech ze serveru, nesmí být během tohoto postupného získávání informací provedena žádná nová změna, například změna pozice automobilu. To by způsobilo, že by jeden z hráčů mohl vidět provedenou aktualizaci dříve než ostatní. Z tohoto důvodu se před všemi změnami musí kolekce s hráči uzamknout prostřednictvím klíčového slova **synchronized**. Poté nastává testování, zda není hra z nějakého důvodu zrušena, pokud ano, byla by herní smyčka ukončena. Dále se prochází postupně všichni hráči ve hře a probíhají veškeré operace potřebné pro tuto hru. Abych urychlil provádění herní logiky odpojení hráči jsou prostřednictvím **continue** přeskočeni. Z důvodu délky kódu uvádím pouze základní kostru smyčky.

```
while (gameRun) { //smycka pro hru
    synchronized (players) { //uzamceni listu s hraci
        if ( isOffline ()) { // zjisteni , zda neni hra zrusena
            break; //ukonceni herni smycky
        }

        for (Player p : players) { //prochazeni hracu
            if (p. isOffline ()) continue; //preskoceni odpojeneho hrace

            //aktualizace pozice a uhlu
            p.getCar().setPositionAndAngle(p.getCar().getAngle2());

            //aktualizace displeje , kvuli testovani kolizi
            p.getDisplay().update(p.getCar().getIncrementY());

            // test kolizi mezi autem a prekazkou
            Collision . TestCollision (map, p.getDisplay(), p.getCar());
        }

        //prochazeni dvojic hracu
        for (int i = 0; i < players.size(); i++) {
            if (players.get(i). isOffline ()) continue; //preskoceni odpojeneho hrace
            for (int j = 0; j < players.size(); j++) {
                if (players.get(j). isOffline () || i == j) continue; //preskoceni odpojeneho hrace

                // test kolizi mezi dvemi auty
                Collision . TestCollisionBetweenCars(players.get(i).getCar(), players.get(j).getCar());
            }
        }
    }
}
```

Výpis 16: Smyčka hry na serveru

7.1.3 Smyčka hráče na serveru

Pro komunikační smyčku s klientem jsem použil také cyklus **while**, který je umístěn v metodě run třídy Player. I zde musí být celý kód smyčky obalen do bloku **try**/**catch** kvůli případným komunikačním problémům. Hlavní činností této smyčky je neustálé načítání a zpracování příchozích příkazů klienta. K získání všech důležitých informací potřebných k vykreslení automobilů na straně klientů slouží příkaz cmdGetPoses, který postupně projde všechny hráče a odešle klientovi informace o pozicích a úhlech všech automobilů. Dalšími důležitými příkazy jsou cmdLeft, cmdRight a cmdRelease, které slouží k ovládání automobilu. V ukázce zpracování několika základních příkazů.

```
while (playerRun) {  //smycka pro komunikaci klienta
    try {
        command = is.read(); //nacteni prikazu

        if (command == CommandClass.cmdOffline) { //testovani prikazu
            isOffline = true; //ulozeni hodnoty do promenne potvrzujici odpojeni hrace
            this.setRunning(false); //nastaveni hodnoty playerRun na false
            break; //ukonceni smycky
        }

        if (command == CommandClass.cmdGetPoses) { //testovani prikazu
            for (Player p : game.getPlayers()) {  //prochazeni hracu ve hre

                if (p.isOffline()) continue; //preskoceni odpojeneho hrace

                dos.writeDouble(p.car.getX()); //odeslani souradnice x
                dos.writeDouble(p.car.getY()); //odeslani souradnice y
                dos.writeDouble(p.car.getAngle()); //odeslani uhlu
            }
        } else if (command == CommandClass.cmdLeftPush) { //testovani prikazu stisknuti doleva

            car.setIncrement(0.09f, 0.79f); //nastaveni hodnot automobilu pro stisknuti doleva

        } else if (command == CommandClass.cmdRightPush) { //testovani prikazu stisknuti
            doprava

            car.setIncrement(-0.09f, 0.79f); //nastaveni hodnot automobilu pro stisknuti doprava

        } else if (command == CommandClass.cmdRelease) { //testovani prikazu uvolenni klavesy

            car.setIncrement(0.09f, 0f); //nastaveni hodnot automobilu pro uvolneni klavesy
        }
        dos.flush(); //vynuceni zapisu neodeslanych dat
    } catch (IOException e1) {
        e1.printStackTrace();
    }
}
```

Výpis 17: Smyčka klienta na serveru

7.2 Klientská část

Komunikace klienta se serverem začíná už v hlavním menu hry, kde se při stisknutí tlačítka pro novou hru naváže spojení se serverem a klientovi se odešle seznam možných map pro hru a dále pak tato komunikace pokračuje během hry. Pro komunikaci jsem implementoval třídu Client, která obsahuje metody potřebné pro veškeré činnosti spojené s komunikací mezi klientem a serverem. Třída je společná pro obě platformy a je tedy umístěna do mé knihovny tříd.

7.2.1 Stažení mapy ze serveru

Jednou z důležitých částí klienta je stažení mapy ze serveru. K tomu slouží metoda loadMap, která ze serveru stáhne mapu v podobě bajtů, ze kterých je později zkonstruován obrázek této mapy pro hru. Tato metoda spočívá v odeslání příkazu cmdLoadMap reprezentující požadavek na stažení mapy, poté je zaslán název mapy, kterou si uživatel vybral z nabídky možných map v menu. Následně je přijatá hodnota ze serveru, která představuje velikost pole bajtů. A poté se konečně příjme pole bajtů pro mapu a ukončí se komunikace uzavřením soketu.

```
public byte[] loadMap(String map) {
    try {
        Socket s = new Socket(ip, port); // vytvoreni soketu na portu a adrese

        os.write(CommandClass.cmdLoadMap); //odeslani pozadavku pro stazeni mapy
        dos.writeUTF(map); //odeslani nazvu mapy
        dos.flush(); // vynuceni zapisu neodeslanych dat

        int len = dis.readInt(); // nacteni velikosti pole
        byte[] array = new byte[len]; // vytvoreni pole na zaklade ziskane velikosti
        dis.readFully(array); // nacteni bytu do pole

        s.close(); // uzavreni soketu
        return array;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

Výpis 18: Metoda loadMap

7.2.2 Získání aktuálních pozic automobilů

Aby mohli klienti vidět ve hře nejen svůj automobil, ale i soupeřův, je nutné stahovat ze serveru aktuální souřadnice, úhly a jiná důležitá data. K uložení všech důležitých informací o automobilech má každý klient seznam pro automobily ve hře, který musí ve své hlavní smyčce aktualizovat. K samotnému stažení nám také slouží třída Client, která obsahuje metodu getPosesAndHp. V této metodě se po odeslání a potvrzení požadavku cmdGetPoses postupně prochází a aktualizují všechny automobily ve hře.

```
public void getPosesAndHp(List<Car> cars) {
    try {
        os.write(CommandClass.cmdGetPoses); //odeslani pozadavku pro stazeni pozic
        dos.flush(); //vynuceni zapisu neodeslanych dat

        for (Car car : cars) { //prochazeni automobilu
            if (dis.readBoolean()) continue;

            car.setX(dis.readDouble()); //nastaveni souradnice x
            car.setY(dis.readDouble()); //nastaveni souradnice y
            car.setAngle(dis.readDouble()); //nastaveni uhlu
            car.setHp(dis.readInt()); //nastaveni HP
            car.setnitroActived(dis.readBoolean()); //nastaveni nitro
            car.setWin(dis.readInt()); //nastaveni informace o vitezstvi
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

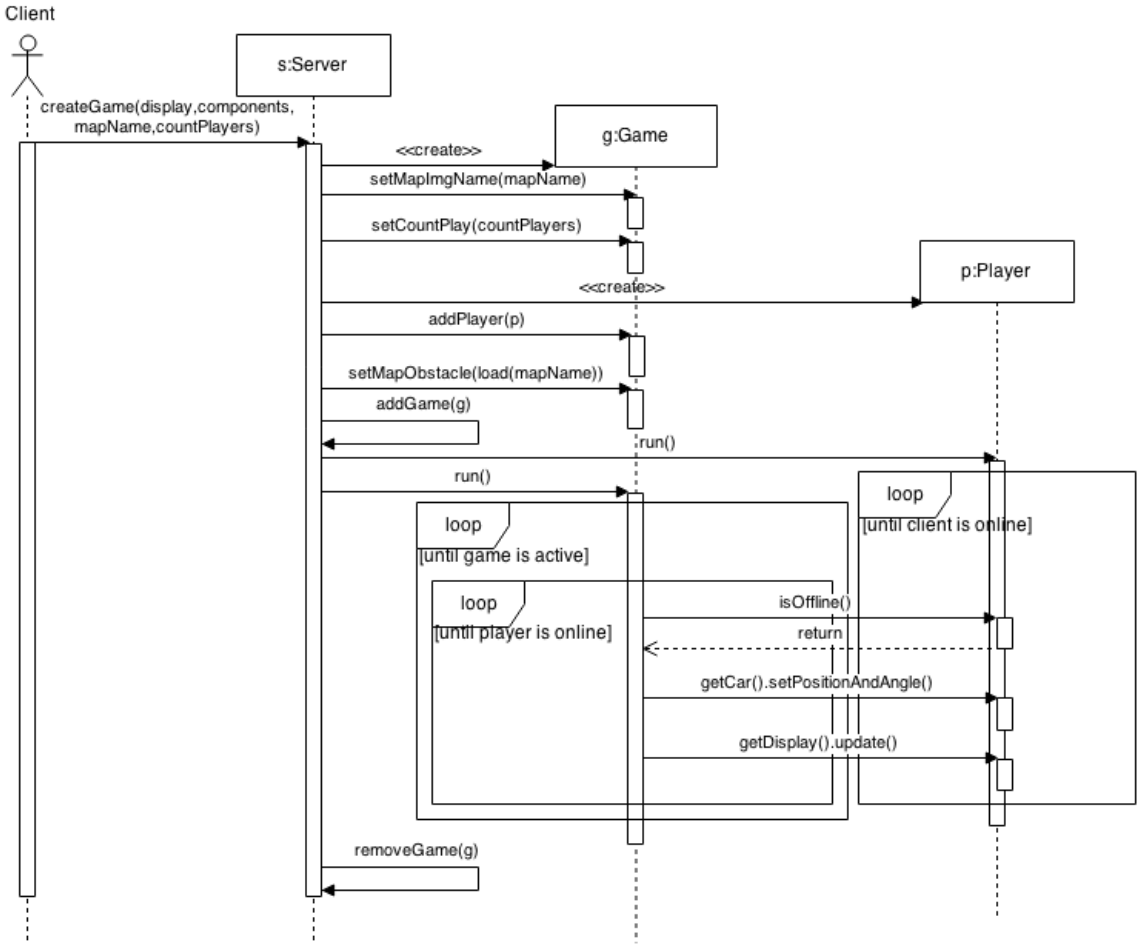
Výpis 19: Metoda getPosesAndHp

7.2.3 Aktualizace hry

Jak jsem již uváděl, v hlavní smyčce každé hry se musí běžně aktualizovat logická a grafická část hry. Jelikož se celá logika hry odehrává na serveru, aktualizace logické části na straně klienta představuje pouze stažení dat ze serveru pro aktualizaci automobilů a aktualizaci displeje. Jelikož se automobil pohybuje vždy ve stejné úrovni nad spodní stranou displeje, aktualizace displeje spočívá v odečtení aktualizované hodnoty souřadnice y od hodnoty této souřadnice před aktualizací. Ukončení hry spočívá v projetí cíle, to znamená porovnání souřadnice y s hodnotou souřadnicí y cíle. Po projetí cíle se aktualizace displeje zastaví, automobil odjede nahoru z viditelné oblasti a tím závod skončil.

```
public void increaseValue(Game myGame, Client client) {
    //ziskani stare hodnoty souradnice y
    double oldY = myGame.getMap().getCars().get(myGame.getIDplayer()).getY();

    myGame.getMap().setAllCars(myGame, client); //aktualizace dat ze serveru

    //ziskani nove hodnoty souradnice y
    double newY = myGame.getMap().getCars().get(myGame.getIDplayer()).getY();

    //overeni zda automobil neprojel cilem
    if (newY < myGame.getMap().getCilObs().getY()) {
        myGame.getDisplay().update(newY − oldY); //aktualizace displeje
    }
}
```

Výpis 20: Metoda increaseValue



Obrázek 14: Sekvenční diagram popisující vytvoření hry

8 Editor map

Informace o tratích máme uloženy ve formátu XML, který by bylo velice složité a zdlouhavé psát ručně. A proto bylo také součástí mé práce vytvořit editor map, který nám bude sloužit k snadnému vytváření těchto tratí. Samotný editor by měl být uživatelsky jednoduchý a mapa by se v něm měla dát vytvořit za relativně krátký čas. Další důležitou vlastností je, aby editor umožňoval uživateli tvořit trať dle svého uvážení a neomezoval ho v základních parametrech, jako je například délka tratě.

8.1 Informace mapy

Samotná trať ve formátu XML musí obsahovat veškeré informace potřebné nejen k vykreslení mapy, ale i informace potřebné k samotné hře. Moje mapa obsahuje informace o prvcích jako jsou: pozadí mapy, překážky v mapě, start závodu, cíl závodu, šířka a výška mapy. Uvedu zde ukázku jednoduché mapy vytvořené pomocí editoru.

```
<?xml version="1.0" encoding="UTF−8" standalone="no"?>
<map height="840" width="399">
    <lines>
        <line  img="start"  x1="−1" x2="399" y1="124"/>
        <line  img="finish"  x1="−1" x2="399" y1="665"/>
    </lines>
    <obstacles>
        <obstacle angle="90.0" img="barriers2Small" x1="111" y1="475" y2="415"/>
        <obstacle angle="90.0" img="barriers2Small" x1="111" y1="525" y2="465"/>
        <obstacle angle="0.0" img="barriers2Small" x1="135" y1="501" y2="441"/>
        <obstacle angle="0.0" img="barriers2Small" x1="85" y1="500" y2="440"/>
    </obstacles>
    <roads>
        <road img="road1Small" x="0" y="0"/>
        <road img="road1Small" x="0" y="20"/>
        <road img="road1Small" x="0" y="40"/>
    </roads>
</map>
```

Výpis 21: Mapa ve formátu XML

8.2 Návrh editoru

Editor map je desktopová aplikace, která pro každou mapu vytváří nejen výše uvedený XML soubor, ale také obrázek dané mapy. Tyto dva soubory jsou při uložení zaslány na server, odkud si je klienti stáhnou. Editor map je rozdělen na dvě hlavní části. V levé části je vyobrazena tvořená mapa, která je rozdělena pomyslnou mřížkou pro přesné vkládání jednotlivých prvků. Tuto mřížku lze skrýt a uživatel má již přímo v editoru pohled na výslednou mapu. A pravá část je panel sloužící k výběru jednotlivých prvků. Tyto prvky můžeme aktivovat kliknutím myši na jeho obrázek, nebo výběrem z textového seznamu, který je tvořen názvy těchto prvků.

První záložka obsahuje pozadí mapy dvou velikostí. První velikost jsou pozadí, které již mají šířku celé mapy a určitou výšku, takže pomocí nich můžeme celkem rychle poskládat mapu o libovolné délce. Druhá velikost jsou pozadí o šířce a výšce jedné buňky mřížky, které slouží spíše na tvoření detailních částí mapy. Druhá záložka prvků obsahuje překážky mapy, které slouží jako svodidla v závodu, nebo například ohraničení určité oblasti v trati. Kde každá překážka má výšku třech buněk mřížky a lze jí libovolně posouvat po mapě kdykoli na ni klikneme a posouváme myší. U překážky lze měnit nejen její polohu, ale i úhel natočení dvojitým kliknutím a posunem myši do stran. Třetí záložka obsahuje start a cíl závodu, které jsou široké přes celou šířku mapy a které můžeme také libovolně po mapě posouvat kliknutím a pohybem myši. Čtvrtá záložka obsahuje přidávání řádků mapy, kdy jeden řádek má výšku jedné buňky mřížky.

Každou trať je možné nejen uložit, ale také znovu načíst do editoru například pro provedení úpravy rozmístění překážek, nebo prodloužení mapy.

8.3 Implementace editoru

Pro vývoj editoru map jsem použil stejný jazyk, jako pro vývoj klientské části pro desktop a server, tedy Java SE verze 1.6. Samotná implementace spočívá vlastně v přesném umísťování prvků do plátna mapy, popřípadě jeho přesouvání a samozřejmě následné ukládání důležitých informací do XML souboru.

8.3.1 Umístění a pohyb prvků mapy

Jelikož jsem i zde pracoval s knihovnou SWT, bylo mi pomocí ní umožněno využít plátno pro vykreslování, které nám dává důležité informace například o tom na jaké pozici x, y se nachází kurzor myši v plátnu. Pokud tedy tuto skutečnost víme, není složité na tuto pozici vykreslit prvek mapy, ať je to pozadí, nebo překážka.

Další věc, kterou jsem musel řešit, je umístění překážky na pozici, kde se již nějaká nachází. Tuto skutečnost jsem využil pro pohyb již umístěných překážek pohybem kurzoru myši po plátnu. Tedy pokud klikneme na již umístěnou překážku v mapě a posouváme kurzorem kamkoli po plátnu, tak se nám mění její souřadnice a vykresluje se stále na nové pozice. Samotné zjištění jestli jsme klikli do překážky je velice jednoduché, stačí použít událost mouseDown a testovat jestli je aktuální pozice x kurzoru větší než levá strana překážky x1 a zároveň menší než pravá strana překážky x2. Ten samý princip platí pro pozici y, tedy musí být větší než horní strana překážky y1 a zároveň menší než spodní strana překážky y2.

```
if (e.x > myObstacle.getX1() && e.x < myObstacle.getX2() && e.y > myObstacle.getY1() && e.y <
    myObstacle.getY2()){

  idBarrier = obstacleList.indexOf(myObstacle);
  changePosition = true;
}
```

Výpis 22: Test kliknutí do překážky

8.3.2 Rotace prvku mapy

Aby nebyly překážky pouze ve vertikální poloze, je potřeba umožnit uživateli jejich rotaci. To jsem vyřešil podobně jako u předchozího případu. Zjištění jestli jsme klikli do konkrétní překážky v mapě je stejné, jediný rozdíl je, že při rotaci jsme použili událost mouseDoubleClick, tedy dvojklik. A to z důvodu rozlišení těchto dvou rozdílných akcí. Rotace pak spočívala v zjištění jestli se kurzor myši pohybuje směrem doprava od překážky, nebo směrem doleva od překážky. V obou případech jsme buď přičítali nebo odečítali určitou hodnotu od úhlu dané překážky.

```
if (actualX > e.x) {
    myObstacle.setAngle(myObstacle.getAngle() − angleIncrement);
}
else if (actualX < e.x) {
    myObstacle.setAngle(myObstacle.getAngle() + angleIncrement);
}

actualX = e.x;
child.redraw();
```

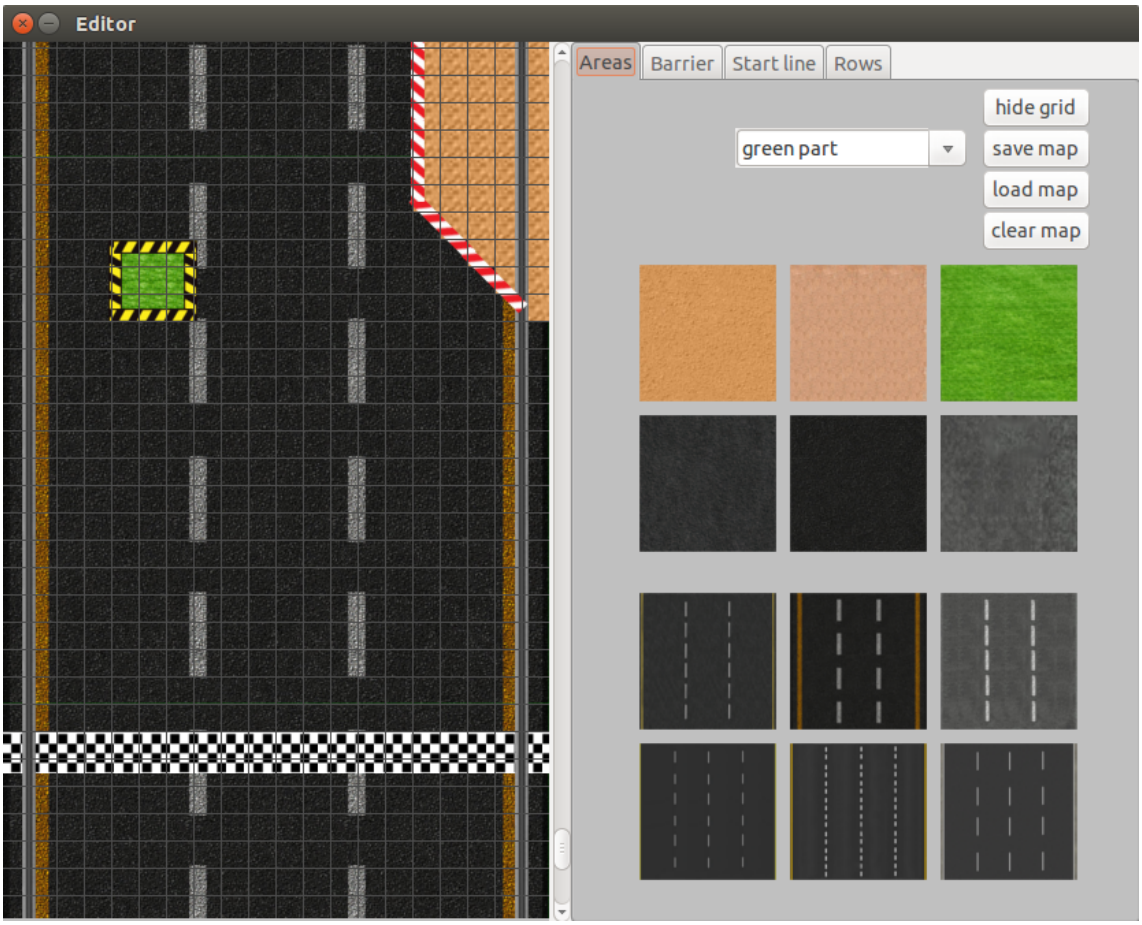
Výpis 23: Změna úhlu překážky

8.3.3 Posun mapy

Aby mohl uživatel vytvářet mapy o libovolné délce, musí mu být umožněn pohyb po mapě pomocí scrollu. Ten je realizován na základě SWT prvku zvaného ScrollBar, neboli posuvník. Který ovládáme myší a on nám poskytuje informaci o tom, kolik je posunut od své výchozí pozice. Na základě této a předchozí hodnoty pozice ScrollBaru se provede jejich rozdíl, který bude představovat posun po plátnu směrem nahoru nebo dolů právě o hodnotu rozdílu.

```
int vSelection = vBar.getSelection();
int destY = −vSelection − origin.y;
child.scroll(0, destY, 0, 0, width, height, false);
origin.y = −vSelection;
```

Výpis 24: Výpočet hodnoty scrollu



Obrázek 15: Editor map s vytvořenou mapou

9 Závěr

Hra na straně serveru je schopna pojmout neomezený počet komunikujících hráčů. Klienti si mohou zvolit mezi několika možnostmi v hlavním menu hry, do kterých patří vytvoření nové hry, připojení se do již vytvořené hry, nebo vylepšení automobilu. Přičemž při volbě vytvoření nové hry je hráči poskytnut seznam map ze serveru, které může hráči poskytnout. Při volbě připojení do vytvořené hry je hráči poskytnut seznam vytvořených her, do kterých se může připojit.

Jedním z cílů této práce bylo také umožnit připojení hráčů ze zařízení s operačním systémem Android, což bylo splněno. Hráči s tímto operačním systémem se mohou připojit libovolně bez ohledu na rozlišení displeje zařízení a mají stejné možnosti pro hru, jako ostatní hráči na PC. Podařilo se také celou logiku hry umístit na stranu serveru, a tím tak oddělit vykonávání veškerých výpočtů od aktualizace grafické části hry, která se vykonává naopak na straně klienta.

Tato práce ukazuje jednu z možností, jak přistupovat ke hře pro více hráčů. Od návrhu serveru, přes návrh hry, až po implementaci těchto částí. Tato práce mi dala pohled nejen na vývoj desktopové aplikace, ale i aplikace pro operační systém Android, díky kterým vím, co je nutné udělat k vytvoření takové hry.

10 Reference

[1] The History Of Video Arcade Games. In: *Bmigaming.com* [online]. [cit. 2015-04-22]. Dostupné z: http://www.bmigaming.com/videogamehistory.htm

[2] POSLUŠNÝ, Radek. Historie počítačových her. In: [online]. 1999 [cit. 2015-04-22]. Dostupné z: http://www.fi.muni.cz/usr/jkucera/pv109/xposlusn.html

[3] TIŠNOVSKÝ, Pavel. Historie vývoje PC her: Hry v textovém režimu. In: *Root.cz* [online]. 2011 [cit. 2015-04-22]. Dostupné z: http://www.root.cz/clanky/historie-vyvoje-pocitacovych-her-3-cast-hry-v-textovem-rezimu/

[4] HOLDER, Dwayne. Need for Speed Retrospective. In: *Gamerant.com* [online]. 2010 [cit. 2015-04-23]. Dostupné z: http://gamerant.com/need-for-speed-retrospective-timeline-hold-51314/

[5] NFS Games. *Ea.com* [online]. [cit. 2015-04-23]. Dostupné z: http://www.ea.com/nfs

[6] MYSLIVEČEK, David. Krátké ohlédnutí za historií Androidu. In: *Svetandroida.cz* [online]. 2013 [cit. 2015-04-22]. Dostupné z: http://www.svetandroida.cz/kratke-ohlednuti-za-historii-androidu-201305

[7] Android 1.5 Platform Highlights. *Developer.android.com* [online]. 2009 [cit. 2015-04-23]. Dostupné z: http://developer.android.com/about/versions/android-1.5-highlights.html

[8] Android 1.6 Platform Highlights. *Developer.android.com* [online]. 2009 [cit. 2015-04-23]. Dostupné z: http://developer.android.com/about/versions/android-1.6-highlights.html

[9] Android 2.0 Platform Highlights. *Developer.android.com* [online]. 2009 [cit. 2015-04-23]. Dostupné z: http://developer.android.com/about/versions/android-2.0-highlights.html

[10] Android 2.2 Platform Highlights. *Developer.android.com* [online]. 2010 [cit. 2015-04-23]. Dostupné z: http://developer.android.com/about/versions/android-2.2-highlights.html

[11] Gingerbread. *Developer.android.com* [online]. 2010 [cit. 2015-04-23]. Dostupné z: http://developer.android.com/about/versions/android-2.3-highlights.html

[12] Honeycomb. *Developer.android.com* [online]. 2011 [cit. 2015-04-23]. Dostupné z: http://developer.android.com/about/versions/android-3.0-highlights.html

[13] Ice Cream Sandwich. *Developer.android.com* [online]. 2011 [cit. 2015-04-23]. Dostupné z: http://developer.android.com/about/versions/android-4.0-highlights.html

[14] Jelly Bean. *Developer.android.com* [online]. 2012 [cit. 2015-04-23]. Dostupné z: http://developer.android.com/about/versions/jelly-bean.html

[15] Android KitKat. *Developer.android.com* [online]. 2013 [cit. 2015-04-23]. Dostupné z: http://developer.android.com/about/versions/kitkat.html

[16] Android Lollipop. *Developer.android.com* [online]. 2014 [cit. 2015-04-23]. Dostupné z: http://developer.android.com/about/versions/lollipop.html

[17] *Indeed.com* [online]. [cit. 2015-04-22]. Dostupné z: www.indeed.com

[18] Oracle Java Embedded. *Oracle.com* [online]. [cit. 2015-04-23]. Dostupné z: http://www.oracle.com/technetwork/java/embedded/overview/index.html

[19] HEISS, Janice. Open Source - Then and Now. In: *Oracle.com* [online]. 2007 [cit. 2015-04-23]. Dostupné z: http://www.oracle.com/technetwork/articles/java/gans-qa-135992.html

[20] Installing the Eclipse Plugin. *Developer.android.com* [online]. [cit. 2015-04-23]. Dostupné z: http://developer.android.com/sdk/installing/installing-adt.html

[21] SEMECKÝ, Vojtěch. Android Studio - nové vývojové prostředí. In: *Zdrojak.cz* [online]. 2013 [cit. 2015-04-23]. Dostupné z: http://www.zdrojak.cz/clanky/android-studio-nove-vyvojove-prostredi/

[22] Smluvní podmínky služby Google Play. *Play.google.com* [online]. 2014 [cit. 2015-04-23]. Dostupné z: https://play.google.com/intl/cs_cz/about/play-terms.html

[23] Distribuční smlouva pro vývojáře Google Play. *Play.google.com* [online]. 2014 [cit. 2015-04-23]. Dostupné z: https://play.google.com/intl/ALL_cz/about/developer-distribution-agreement.html

[24] SUNDSTED, Todd. Introduction to the AWT. In: *Javaworld.com* [online]. 1996 [cit. 2015-04-23]. Dostupné z: http://www.javaworld.com/article/2077188/core-java/introduction-to-the-awt.html

[25] Swing. In: *Java.wikia.com* [online]. [cit. 2015-04-23]. Dostupné z: http://java.wikia.com/wiki/Swing

[26] SWT - Tutorial. In: *Vogella.com* [online]. 2013 [cit. 2015-04-23]. Dostupné z: http://www.vogella.com/tutorials/SWT/article.html

[27] SWT: The Standard Widget Toolkit. *Eclipse.org* [online]. 2014 [cit. 2015-04-23]. Dostupné z: http://www.eclipse.org/swt/

[28] Java Network Launch Protocol. *Docs.oracle.com* [online]. [cit. 2015-04-23]. Dostupné z: https://docs.oracle.com/javase/tutorial/deployment/deploymentInDepth/jnlp.html

[29] Java Web Start Overview. *Oracle.com* [online]. [cit. 2015-04-23]. Dostupné z: http://www.oracle.com/technetwork/java/javase/overview-137531.html

[30] Supporting Multiple Screens. *Developer.android.com* [online]. [cit. 2015-04-23]. Dostupné z: http://developer.android.com/guide/practices/screens_support.html

[31] SOJKA, Eduard, Martin NĚMEC a Tomáš FABIÁN. *Matematické základy počítačové grafiky* [online]. Ostrava, 2011, s. 1-30 [cit. 2015-04-24]. Dostupné z: http://mrl.cs.vsb.cz/people/sojka/pg/mzpg.pdf

A Příloha na CD

A.1 Zdrojové soubory

- klientská aplikace pro desktop (desktopApp)
- klientská aplikace pro Android (androidApp)
- knihovna společných tříd (gameLib)
- serverová aplikace (server)
- editor map (mapEditor)

A.2 Spustitelné soubory (jar)

- klientská aplikace pro desktop (Highway Racing)

A.3 Mapy pro hru (xml, png)

- fastRace
- longRace

A.4 Uživatelská příručka (pdf)

- popis ovládání v menu hry Highway Racing (6 stran)